TECHNISCHE UNIVERSITÄT
CAROLO-WILHELMINA
ZU BRAUNSCHWEIG

Interactive Latency-Sensitive Applications in Mobile Ad-hoc Networks

Von der Carl-Friedrich-Gauß-Fakultät
Technische Universität Carolo-Wilhelmina zu Braunschweig
zur Erlangung des Grades Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation von
Oliver Wellnitz, geb. 30.07.1973 in Celle.

Eingereicht am: 27.09.2011

Mündliche Prüfung am: 09.02.2012

Referent: Prof. Dr.-Ing. L. Wolf

Korreferent: Prof. Dr. B. Plattner (2012)

Abstract

In this thesis we discuss the challenges that latency-sensitive interactive applications face in mobile ad-hoc networks. By using multi-player games as an example, we argue that the traditional client-server architecture is unsuitable for this new environment. We consequently create a novel communication architecture as well as quality of service mechanisms that can support the network requirements of such applications in mobile environments. By using a number of distributed zone servers that are selected and managed dynamically by our server selection algorithm, we provide a scalable approach that offers the necessary redundancy. Furthermore, we propose additional quality of service mechanisms to reduce latency and packet loss for interactive applications. We evaluate our approach through network simulation and realistic mobile gaming scenarios. The performance of our evaluation is checked against real-world measurements.

Kurzfassung

In dieser Arbeit werden die Probleme und Herausforderungen von latenz-kritischen interactiven Computeranwendungen in mobilen Ad-hoc Netzen untersucht. Am Beispiel von Mehrbenutzercomputerspielen zeigen wir, dass traditionelle Client-Server Architekturen für diese neuen Umgebungen ungeeignet sind. Im Rahmen dieser Arbeit wird daher eine neue Kommunikationsarchitektur sowie verschiedene Mechanismen zur Erhöhung der Dienstgüte vorgeschlagen. Mit Hilfe von Zonenserver, die durch den Serverauswahlalgorithmus ausgesucht und verwaltet werden zeigen wir einen Ansatz auf, der sowohl bezüglich der Netzgröße skalierbar ist als auch die notwendige Redundanz bereitstellt. Wir zeigen die Funktionalität und die Leistung unseres Ansatzes mit Hilfe von Netzsimulationen bei denen realistische Szenarien für mobiles Spielen simuliert werden. Der hierbei benutze Netzsimulator wurde dafür auf Basis von eigenen Messungen verbessert und für das jeweilige Szenario passend eingestellt.

Contents

1. Introduction

One of the most important challenges in data communication today is the ongoing amalgamation of computer networking and user mobility. With the introduction of high speed data radio networks more than a decade ago, an ever increasing number of users today use the Internet wirelessly. The advantages for the user are obvious. Equipped with a wireless network adapter and a battery as power supply, the user's mobility is no longer limited by cables. Additionally, energy-saving mechanisms in hard- and software as well as improved battery capacities allow operating times of six or more hours for mobile computers without any need for cables. Surfing the web and sitting comfortably on the terrace or in the garden is nowadays considered standard at home as more and more people rely on wireless networks. In the business area, company meeting rooms and hotels are often equipped to allow wireless Internet access. In addition to the newly achieved freedom of movement, wireless networking also saves time and effort of deploying a wired network. On the road, these wireless local area networks (WLANs) are augmented by mobile telephony networks that support wide area coverage but with lower performance. Initial tests with WLAN on trains and planes have been made but are not widely deployed. All mentioned wireless technologies have in common that they need infrastructure in the form of a central point, a base station, through which all traffic is passed on. Even for mobile devices that are local neighbours, communication is handled through a base station. They are the required infrastructure for a wireless network. Ad-hoc networks follow the opposite paradigm and allow mobile devices to directly communicate between each other providing a number of advantages. They can use the wireless medium more efficiently than an infrastructure network because data transmissions do not have to be sent back and forth to a base station. They also allow higher data rates if sender and receiver are close together. And finally, they are more cost effective as they do not require any network infrastructure to control and coordinate traffic. On the downside, managing a network of mobile hosts without a stationary intermediary like a base station is more difficult.

In the area of applications, the constantly advancing progress in networking technologies and their widespread use has led to the development of programs that allow people to interact with each other directly. Computer games, telephony and chat applications,

as well as playing music together over the Internet are only some examples. Interactive applications often have different network requirements than other kinds of applications focusing more on latency and jitter than throughput. A maximum latency between 100 and 150 ms are common for interactive applications and most programs can deal with five percent of lost packets. Using existing latency-sensitive and interactive applications in mobile networks is a challenging task, first of all due to that fact that mobile network capacity is not limitless. In the Internet, performance problems are usually solved through over-provisioning by either using a faster network technology or multiple cables to increase the network capacity. Both strategies cannot be applied to mobile networks. Here, the wireless medium cannot be extended and existing transmission rates are significantly lower than those used in the backbone of the Internet. Secondly, in wireless networks one communication partner (when using a fixed base station infrastructure) or even both communication partners (in a mobile ad-hoc network) could move around which has a significant effect on network performance. Finally, a mobile ad-hoc network lacks infrastructure components which can coordinate and manage data traffic. As a consequence, it is still an open question how well latency sensitive applications will run in these networks and what adjustments are necessary to optimise performance.

In the remainder of this introduction, we will take a closer look at the challenges interactive applications face in mobile ad-hoc networks. We then present our main three goals for this thesis and briefly outline suitable approaches to achieve them. We conclude this chapter with an outline of this thesis and some terms and definitions that we will use throughout this thesis.

1.1. Challenges

Every network application has a variety of requirements regarding the availability and performance of resources like CPU, memory, hard disk, network and others. From the network point of view, these requirements often include latency, latency variation, packet loss, and throughput. Today, existing applications are often designed primarily to work within a local network and/or the Internet. Both networks offer high bandwidth combined with low bit error rates and low delay so that the network requirements of most application including interactive ones can be met easily. Furthermore, local networks and the Internet offer reliable communication services so that applications are able to use a single centralized server to help coordinate data between applications on multiple machines. From a network perspective, mobile and wireless networks have a different set of performance characteristics than wired networks. Hence, it is

doubtful that existing approaches perform similarly in these new environments.

Throughout the research community several approaches have been proposed to communicate directly and wirelessly between mobile nodes without the need for costly infrastructure components. Some ideas even support mobility for intermediate systems as well. The simplest solution is direct communication between two mobile partners. A more complex approach considers mobile intermediate nodes to form a mobile multi-hop ad-hoc network or MANET. Here, multiple mobile nodes can communicate with each other and network traffic is forwarded by any node through the network. In MANETs, the differences between end host and network infrastructure disappear as every mobile node performs the role of host and intermediate system at the same time, creating a mobile network of equals.

Wireless networks are not as reliable as compared to traditional wired networks. Their problems come from using of radio communication that, when used in combination with user mobility, creates a number of new problems. First and most important, radio communication is generally open to outside interference either in the form of obstacles blocking the line of sight between two stations or electromagnetic noise both of which have a negative impact on network performance. In most cases such interference is also unpredictable. Secondly, user mobility influences network quality and connectivity because the signal strength of a radio transmission is affected quadratically by the distance between communicating partners. Together these problems can result in fluctuating network performance and packet loss in a mobile ad-hoc network which makes it difficult for existing applications to work in such an environment.

The aim of this thesis is to determine if and how today's interactive and latency-sensitive applications can be used in mobile ad-hoc networks. For this, we introduce the following three topics on which we will focus in this work and which are described below:

- Our zone server approach as a novel communication architecture for MANETs
- Analysing latency and providing quality of service for wireless data communication
- Improving realism in the simulation of latency in mobile ad-hoc networks

Existing interactive and latency-sensitive applications in the Internet often use a client-server approach. But due to user mobility and radio interferences, a single point of failure is not acceptable in mobile ad-hoc networks. The opposite approach, a fully-distributed peer-to-peer architecture with many clients may overload the mobile network and/or the computing power of individual mobile clients. Hence, we have developed a new zone server architecture which combines both approaches and provides a

suitable communication environment for latency-sensitive applications in mobile ad-hoc networks. This architecture is specifically tailored to support mobility and is scalable beyond the range of a single wireless collision domain. A novel zone server selection algorithm provides all necessary information about servers and neighbouring nodes and offers quick start-up times with very low overhead.

In wired networks the necessary performance is ensured by over-provisioning the network which keeps the network queues short and prevents overloading. As a result support for quality of service is not widely employed in the Internet. In radio networks, over-provisioning is not feasible today as even high speed wireless networks offer only a portion of bandwidth that is available for their wired counterparts. In this thesis, we analyse wireless LAN to determine the factors through which latency originates. By using our modified WLAN driver and real world measurements, we can gather detailed and accurately time-stamped information about transmissions. By using a new approach to correlated this data we can break down delays in its most basic components. Based on this information we are able to determine the maximum number of mobile nodes running latency-sensitive applications that this technology can support today. We also take a look at quality of service methods which improve performance for interactive applications. We study the effects of various methods in the area of medium access control, routing and traffic management to find the best improvements for latency-sensitive applications in situations with and without additional background traffic.

Finally, we evaluate our approaches using a series of network simulations. However, the current simulation tools used by the research community often focuses on network throughput rather than latency. Moreover, they are regularly used in a more common sense without focusing on specific scenarios. In order to provide a more realistic evaluation we use real-world measurements to study the usability of existing network simulation tools for latency-sensitive applications. We show that our improved network simulator for mobile ad-hoc networks provides a more accurate reproduction of delay and packet loss. Together with a number of selected scenarios, we can show the performance of our zone server architecture as well as the limits for latency-sensitive applications in WLAN.

1.2. Outline

The remainder of this thesis is structured as follows:

In Chapter 2, we take a look at the network requirements of latency-sensitive applications and especially look into multi-player games as our example for this group of

applications. We also discuss the current state of art in mobile gaming and present three different scenarios in which mobile multi-player games are likely to be played.

Chapter 3 presents background and related work on mobile ad-hoc networks and existing game server architectures. We will also briefly discuss wireless LAN as an example of an ad-hoc link layer protocol. Finally, we present our zone server architecture which is especially designed for distributed applications to work in mobile ad-hoc networks.

Chapter 4 discusses our server selection algorithm which discovers neighbouring nodes and chooses suitable application servers. We present the general design of the algorithm and explain its implementation in detail. We also provide a brief qualitative evaluation of the algorithm

Chapter 5 deals with quality of service in mobile ad-hoc networks. Here, we propose improvements for latency-sensitive applications on the MAC, the networking and the application layer. In a pre-evaluation, we will determine the effect that each change has on latency-sensitive applications as well as any burden it might put on other traffic in the network.

The evaluation is presented in Chapter 6. Here, we compare real-world measurements from a simple scenario to assess the quality of our simulation environment. Based on these initial results, we select a suitable network simulator to ensure a realistic evaluation. Thereafter, we we use scenarios and application traffic characteristics discussed earlier in Chapter 2 to evaluate our server selection algorithm. Lastly, we analyze application traffic to find out under which conditions the requirements for our latency-sensitive application can be achieved.

Finally, Chapter 7 concludes this thesis and summarizes our contribution. We will also point out open issues and ideas for future work

1.3. Terms and Definitions

Some terms used in computer science and especially in networking are not well-defined or have multiple meanings that serve different purposes and are valid from their respective points of view only. In this section, we would like to clarify those ambiguities. We also introduce specific terminology when referring to mobile nodes in this thesis which we would like to clarify in advance. We therefore define the following terms:

Bandwidth

In computer science, the term 'bandwidth' is often used incorrectly meaning

maximum achievable gross throughput rather than a frequency range. In this work, we use bandwidth in relation to the technical characteristics for a wireless technology. Thus, we define bandwidth to be the gross throughput that a technology can achieve under the described conditions.

Delay Variation / Latency variation

The term 'jitter' is used frequently to describe the variation of latency, however, this term itself is not well-defined. Hence, we avoid using the term 'jitter' but will use the term 'delay variation' as defined in [1] which is the average of the absolute differences in delay between two consecutive packets. With n packets and the delay of packet x being d_x, the delay variation is defined as follows:

$$d_{var} = \frac{1}{n-1} \times \sum_{x=1}^{n-1} |d_x - d_{x+1}|$$

We use the term 'latency variation' in the same context. In addition, we will use variance and standard deviation to describe how latency is distributed over time.

The following terms are used in relation with our zone server selection algorithm:

Server node

A server node is a mobile node that is determined to be a server by a another node's instance of the server selection algorithm and has agreed to this role by changing its status within its own server selection algorithm instance accordingly.

Neighbouring nodes

Two mobile nodes that run an instance of an interactive latency-sensitive application and that are able to communicate directly with each other are called neighbouring nodes. Communication between neighbouring nodes does not require forwarding of data by another node.

Single nodes (Singles)

A mobile node that runs an instance of an interactive latency-sensitive application that has no neighbouring nodes is called single node.

Background nodes

A background node is a mobile node that operates within the mobile ad-hoc network but does not run an instance of an interactive latency-sensitive application. Like any other node in the mobile ad-hoc network, background nodes forward data according to the common routing protocol. However, they are unable to

take part or support any server selection. Background nodes may or may not create network traffic of their own by running other applications.

Acknowledgments

I like to acknowledge the work of Dirk Budke and Edgar Liptay who wrote their master and diploma theses under my supervision. Both studies provided valuable results which were helpful in wirting this thesis. I also like to thank my former colleagues at the Institut für Betriebssysteme und Rechnerverbund, TU Braunschweig for many fruitful discussions and Prof. Lars Wolf for his valuable comments.

2. Applications & Scenarios

In this chapter, we start by examining latency-sensitive interactive applications and discuss their network requirements in general. We take an in-depth look at multi-player games as an example of such applications and provide a classification into different game types each of which have distinct demands from the network. Furthermore, we present the current state of the art in mobile gaming and give an outline and rationale for games in mobile ad-hoc networks. We move on to discuss plausible scenarios in which mobile games are likely to be seen in the future. We conclude this chapter with evaluation criteria for mobile gaming in different scenarios which will be picked up again later on in the evaluation in Chapter 6. Finally, we present a brief summary.

2.1. Latency-sensitive applications

Applications have been classified and differentiated in many different ways. They can work synchronously or asynchronously, locally or in a distributed system, in a discrete or continuous fashion. In this section, we take a look at network applications that require a certain timeliness in their interaction with the user because the user interacts directly with a remote application or with another user through a network . In order to ensure a good user experience, such an application should give a reaction to any user action within an appropriate time frame.

For the purpose of this thesis, we define such distributed and interactive applications through the following three peculiarities:

(1) The application provisions a service directly to a user (in contrast to providing it to another machine).

(2) The user expects a timely reaction to his actions.

(3) Multiple applications or application instances run and communicate in a networked environment.

From a network point of view, a low end-to-end latency is the most prominent task for providing the user with a timely response. Therefore, we use the term *latency-sensitive*

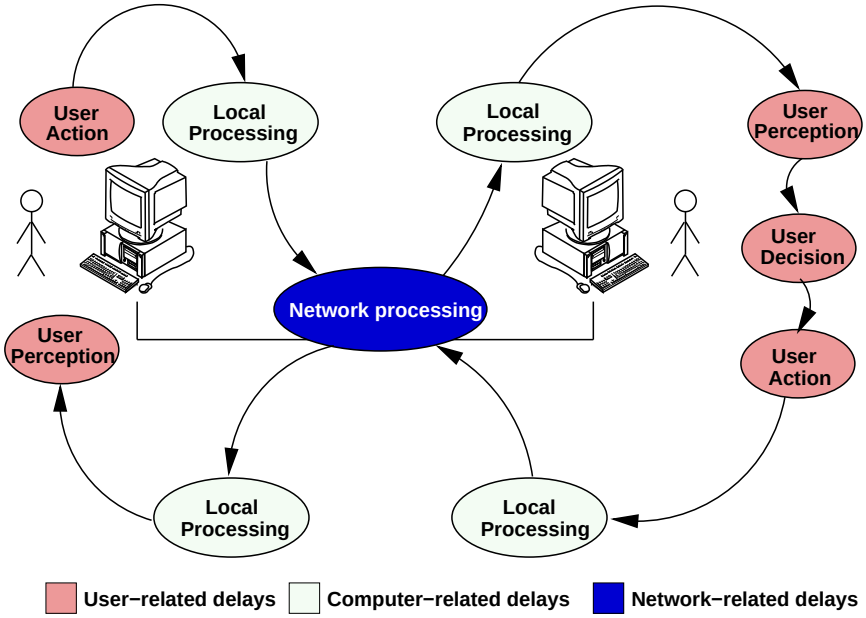


Figure 2.1.: End-to-end delay breakdown for an latency-sensitive application

application to describe these types of applications. Some examples of latency-sensitive applications are multi-player games, video conferencing or interactive video streaming in which the user can influence playback through a control channel. Another example is network music performance.

In Figure 2.1 we take a look at an example of an latency-sensitive application in which two users communicate directly with each other through their applications. We follow the action of the left user until a response to his action is perceived and split down the individual delays in the end-to-end communication process. In this figure, the red areas represent user-related actions including the decision process that leads to an action. Though we have used a very basic and simplified perception model for our users, its task is easy to comprehend. The green fields stand for the time needed by the users' computers to process and distribute data between the user and the network which includes tasks such as data processing by the application or the rendering of data for the user. Finally, the processing in the network includes transmission, propagation, queueing and processing delays inside the network itself. For reasons of simplicity, Figure 2.1 shows only the interaction of two users. An example with more than two users or additional infrastructure components like, for example a server, can be created similarly. Obviously, a server component would increase computer processing and network delays only. For the remainder of this work, we will focus on efficient network processing.

2.2. Network Requirements

In related work, various measurements and traffic analyses for latency-sensitive applications can be found from which we can identify general basic network requirements for interactive latency-sensitive applications. We will present a general overview by showing some of these results below using our aforementioned applications as examples. Almost all latency-sensitive applications use simple transport protocols such as the User Datagram Protocol (UDP). UDP does not use any rate control scheme or retransmits packets if they are lost during transmission. The advantage of this approach is that the application does not have to cope with such features that general transmission protocols provide that are not required by a specific application. Instead, it leaves it up to the application to implement methods like automatic repeat request (ARQ) or others in its application protocol if required.

2.2.1. Latency requirements

If we take a look at latency requirements for our applications, several examples in related work can be found. For multi-player games, Begbederet al. [2] concluded that round trip times between 75 ms and 150 ms are advisable for fast-paced multi-player games. In [3], we have shown that delay requirements for multi-player games are very game-dependent and certain games require round trip times to be well below 150 ms in order for the player to have a pleasant gaming experience. The ITU-T recommendation G.114[4] also states that by a one-way delay of less than 150 ms, “most applications will experience essentially transparent interactivity”. Keeping in mind that, e.g., VoIP codecs also introduce delays in the order of 50 ms for GSM or 80 ms for G.723.1, VoIP applications and multi-player games have similar requirements on network latency. Another application example is network music performance where musicians perform together while they are at separate physical locations. This type of application generally requires a maximum one-way latency of 100 ms. Still, special requirements with a maximum of 20 ms are possible as well[5].

2.2.2. Throughput Requirements

Also important, but often to a lesser extent for latency-sensitive applications is network throughput. Generally, the throughput requirements for interactive applications is small with typical bandwidth requirements being in the tens or low hundreds of kBits/s. To achieve low delay, latency-sensitive applications often send data in small

packets of about 20 – 250 bytes over the network to keep packetisation and transmission delays to a minimum. The required data rate is then reached by sending a high number of packets per second. For example, the action multi-player game 'Counter Strike' sends and receives approx. 20 packets of 100 bytes each per second in each direction. For VoIP, data is send and received in 160 bytes packets at a rate of 50 packets per second when using the standard G.711 codec. For the optimized G.723 codec, the packet size gets down to 24 bytes per packet but data is still send at a rate of 34 packets per second[6]. In network music performance applications rates of 300 packets/s or more are not uncommon.

2.2.3. Packet Loss Requirements

The loss of packets during transmission also effects the communication between applications. But because data is sent in short time intervals, a certain loss of packets is often acceptable because its effect can be hidden from the user and the lost packets can be replaced shortly thereafter by arriving new data. Also application-specific compensation techniques or concealment strategies are used to cope with packet loss and/or delay variation. In VoIP applications, lost data is simply replaced by replaying the last received packet to the user. Such a single packet only covers a brief time interval, e.g. 20 ms in case of the G.711 codec, and because the same sound fragment is replayed, pitch and volume is kept at similar levels. Hence, a single lost packet is barely noticeable by the user. According to [7], VoIP applications are „tolerant to about five percent packet loss averaged across an entire call". For video streaming the situation is somewhat different. With the most prominent video compression methods such as MPEG 2, MPEG 4 or H.264, video frames are encoded by using a differential and motion-compensated coding scheme. Due to this, the impact of a single lost packet may not be restricted to a single frame. Hence, depending on the kind of packet and the number lost, video degradation usually occurs when packets are lost. Error recovery strategies for video streaming therefore employ the retransmissions of missing data if there is still enough time left before a video frame needs to be display to the user. For multi-player games such as 'Counter Strike', the players position is distributed regularly to the game clients. This position update often includes not only the current position but also information about the direction and speed of movement (movement vector). The game logic then applies a game-specific mathematical function to interpolate movement between the two or more consecutive positions updates of a player (dead reckoning). This function often includes a prediction mechanism that can deal with lost or late packets as it simulates the behaviour of the standard player. Because the position of every player is computed locally, the user often does not notice

packet loss if below a certain threshold. For games, the term 'latency compensation technique' is often used for a position update algorithm in related work. Acceptable packet loss for multi-player games is in the range of 3 to 5 %[2, 8].

2.2.4. Requirements Regarding Variation of Latency

Finally, the variation of latency can cause additional problems. Our example applications prefer a steady stream of data because they have to display a picture every 1/25th second, play a sound every 20 ms or update a player position every 50 ms. In the case of high variation in latency, this can cause packets arrive too late or too early. Packets that arrive too late are often of little use to the application. If they arrive too early, they have to be buffered by the application until they are needed. Using such a 'jitter buffer' is a standard method against delay variation. It is also useful for reordering packets if packets get out of order during transmission. On the downside, jitter buffer generally increase packet latencies.

2.3. Multi-Player Games

In the previous section, we have looked at latency-sensitive applications in general. From our three examples, we have chosen one application, multi-player games, which we will discuss in detail in this section. Multi-player games are computer-based games where people can play or compete with one another. In contrast to multi-player games, single-player games allow to play only with or against computer-generated figures. During the last two decades and with the advancement of computer networks, like the Internet, multi-player games have become more and more popular. Nowadays, the majority of computer games is specifically designed for direct player interactions.

The first computer games were invented in the forties and fifties of the last century. Thomas T. Goldsmith Jr. and Estle Ray Mann filed a patent for their 'Cathode-Ray Tube Amusement Device'[9] in December, 1947. Although contested, William Higinbotham's *Tennis for Two* (1958) is regarded as the first computer game. It also was the first multi-player game in which two players contested with one another. Later in the 1980s with the success of the Commodore 64, the Atari ST and later the IBM PC, computers became generally available and affordable to the public. In the first era of online gaming, multi-user dialup bulletin board systems were used to host computer games in which more than two players could play. Text adventure and role playing games were often found in these times. Later in the decade, video game consoles such

as Sega's Master System or Nintendo Entertainment System (NES) revived the struggling console market that had existed since the early 70s. Games like Super Mario Bros. are examples of that period. The next revolution came in the late 90s and the early years of our century with the breakthrough of widespread Internet access. In later years, broadband technologies like digital subscriber lines (DSL) and digital cable networks added to this success. Most newer games now offer not only a single-player mode where one can compete with or against other computer-controlled players but a multi-player mode that allows multiple human players to play with or against each other. Nowadays, the gaming market is well established. In 2006, Nintendo released its video game console Wii which offers a wireless controller that uses infrared sensors and an accelerometer to sense its position and movement vector thus further increasing the player's immersion into the game. With more than 30 million sales worldwide, Nintendo's Wii has been a commercial success.It is presumed that the next generation of computer games will either be mobile games specifically designed for portable devices, immersive mixed reality games or a combination thereof. More on the history of computer games in general or multi-player games can be found in [10] and [11].

From a commercial point of view, computer and video games are a huge market. In Germany and according to the federal association of interactive entertainment software (BIU, Bundesverband Interaktive Unterhaltungssoftware e.V.), the total turnover in 2006 for computer and video games was 1.13 billion Euros. Compared to 2005, this market increased significantly by 7.4 percent. This growth was particularly carried by video game sales which increased by 13 percent. In the United States, computer and video game software sales grew six percent to $9.5 billion in 2007. According to the entertainment software association (ESA), 69% of heads of US households play computer or video games[12]. Contrary to the common belief that personal computers are the stronghold of the computer gaming industry, it is in fact game consoles, that are mainly responsible for the commercial success of computer games.

Games are often categorized in various genres that characterize type, pace, and sometimes even the contents of the game. For example, Quake and Unreal Tournament are first-person shooter games. The first person refers to the perspective of the player during the game in contrast to a third-person or bird's-eye view. Like the name indicates, the goal of first-person shooter games is to do combat against other players or computer-controlled enemies. Another popular genre are massively multi-player games like World of Warcraft or Eve Online. Here, persistent virtual worlds with tens of thousands concurrent users play together. Other games types include strategy games (e.g. Command and Conquer), role playing games (e.g. Diablo), or sport games (e.g. Need for Speed). Additionally, more traditional card or board games often also have an equivalent computer game.

In related work, one can find many different approaches on how games can be divided in different categories. In his PhD dissertation, Henderson categorizes games into four groups: First-person shooters (FPS), real-time strategy games (RTS), massively multi-player online role playing games (MMORPG), and non-realtime games. He also states that "the FPS, RTS and MMORPG genres account for the majority of networked games"[10]. Henderson used the game architecture, the number of servers and players per server, and the persistence of the game world as parameters for his classification. In [11], Armitage, Claypool and Branch present four different game genres, namely FPS games, Massively Multi-player games, RTS games and Sports games.

In this thesis, we take a different approach than related work to categorize games by looking at multi-player games from the player's point of view. We have chosen this method because we believe that the users' experience is the most prominent factor when looking at an interactive application. Figure 2.2 shows our multi-player game categories. We divide games into three general categories. Firstly, games that can be played continuously and concurrently by multiple players. Games that are played in a round-robin or any other round-based fashion make up the second category. Finally, hybrid games use a combination of continuous and round-based gameplay during different phases of the game. Many of today's multi-player games fall in the first category. They offer two basic different approaches on how the user interacts with the game. Either the user has direct control over his avatar or item in the game or he can give orders to the game, thus controlling the game indirectly. With both game types the player perceives the game differently because of the two distinct methods of influencing the game.

Below, we will discuss all mentioned game categories in detail and give some examples of current games of that type. Because we have classified multi-player games concerning user perception, we are able to conclude unique network requirements for each category.

2.3.1. Games with directly controlled avatars

The first category describes games in which the player has direct influence over his avatar or character. This means that e.g. the player can move his virtual character by giving commands like forward, left, right, stop and so on. Any command or keystroke is transmitted to the game server which checks it for validity and reports back if such a move is allowed or not. Games with directly controlled avatars often use a first-person view where players sees virtually through the eyes of their avatars. Another possibility is a bird's eye view where players see their avatars from above. Examples of directly controlled games are first-person shooters or some sports games like car racing.

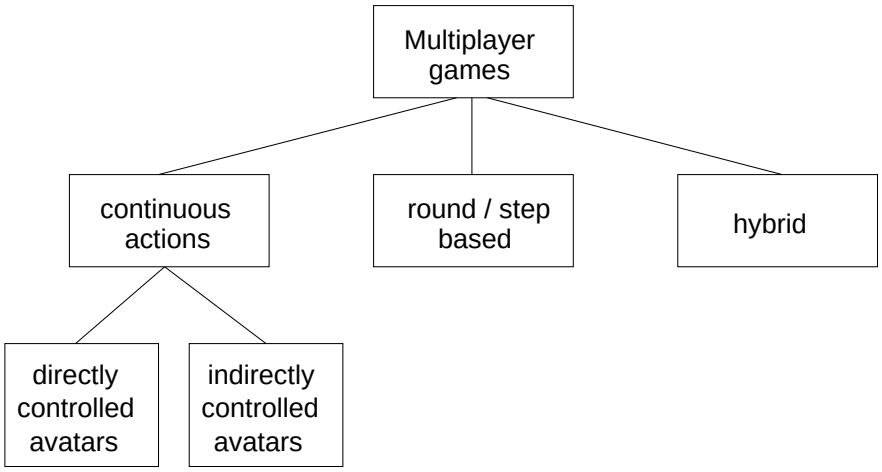


Figure 2.2.: Multi-player game categories

With directly controlled games, players expect almost instantaneous reactions of their avatars in according to their commands. For example, imagine the driver of a racing car who wants to make a turn at high speed. At 100 km/h, a car travels one meter every 36 ms. Achieving synchronism with such high accuracy is quite difficult when using a wireless network when you consider that the action has to be transmitted to the server, processed by the server, and then sent back to the player. Thus, directly controlled games are generally very challenging when it comes to their network requirements. Often latency hiding techniques are used to compensate for higher, varying delay and even packet loss. In the case of lost or late packets, dead reckoning techniques are required to predict the behaviour of others players based on basic rules that describe the expected behaviour of a player. For example, on a straight road in a car racing game, a common prediction is that the player avoids hitting other cars and keeps his car in the middle of the road. Dead reckoning algorithms are game or genre specific which often results in different in-game behaviour in the case of high latency or packet loss.

Games with directly controlled avatars have been widely discussed in related work. Färber[8] looked at measurements from a 36 hour LAN party and determined the specific traffic characteristics for the first person shooter “Counter Strike”. He found that a Counter Strike client on average sends and receives approx. 20 packets per second each utilizing a bandwidth of 4 kBytes/s total during the game.

Beigbeder et al. [2] looked at the effects of loss and latency on “Unreal Tournament 2003”, another first-person shooter. They noticed that users rarely even notice packet loss at or below 5 % but that shooting accuracy is greatly effected by 75–100 ms la-

tency. Comments from players during their tests state that latencies as low as 100 ms were noticeable and players found latencies of 200 ms to be annoying. The authors suggest to keep delay below 150 ms and packet loss below 3 % to achieve an enjoyable user experience.

Armitage[13] found similar results for Quake III where players seem to prefer game servers with low latency. By looking at server logfiles, he suggests that players tolerate delays of up to 150 – 180 ms before using a different server.

Quak et al. [14] concluded from their experiments with Unreal Tournament that players notice impairment with delays above 60 ms. They also mention that jitter does not play a prominent role with that particular first-person shooter game.

Nichols and Claypool[15] analyzed the effects of latency on Madden NFL Football which remains playable even with delays of up to 500 ms. They suggest that Madden NFL uses a "prediction of the round trip time to delay user inputs in an attempt to compensate for any latency effects across both players". This method, however, fails when one-way latencies are asymmetric because in that case simply dividing the round-trip delay in half gives inaccurate results. Again, with bandwidth utilization below 20 kB/s, Madden NFL does not require large link capacities.

Wattimena et al[16] have investigated Quake IV and found out that latency and jitter have both a significant impact with a regression coefficient $R = 0.98$ on the perceived game quality and the results of the game. Especially the introduction of jitter resulted in a large negative effect on the subjective game quality. In [3], we analyzed different kind of games and found that latency and jitter have different effects on distinct games. For example, with Counter Strike, the introduction of jitter was the most disturbing influence, where for Unreal Tournament 2004 only latency was the important factor. We concluded that for games of this category the design of the network protocol and the latency compensation techniques used in these games mainly influence the behaviour of a game in different network situations.

We conclude from related work that directly controlled games place high requirements on the network in order for the game to be playable and fun to the user. Latency and jitter are the most important factors for these kind of game. Packet loss does not play a prominent role and the loss of some packets of the game traffic hardly affects the game play if the overall packet loss stays below five percent. Finally, bandwidth is of no concern for directly controlled games in the Internet when we look at the player' side[1]. Games with directly controlled avatars often use UDP as transport protocol because the

[1] Bandwidth may be of concern for a game server. However, for the moment, we are focusing on the individual player and will discuss the server side later on.

high network requirements do not allow for retransmissions or exponential backoffs which are present in reliable protocols such as TCP.

2.3.2. Games with indirectly controlled avatars

The second category encompasses games where avatars are not under the direct control of the player. With this game type, a player does not steer an avatar itself but rather defines goals that the avatar tries to achieve on its own. For example, a player can direct an avatar to move to a certain destination, attack another player's avatar or chop wood until further notice. The game logic then decides on its own how to achieve that goal. In the case a movement order was given, the computer calculates the best way for the avatar to take in order to reach its destination. The calculated path might not always be a straight line but includes detours to get around impassable terrain or other obstacles. Often with games of this category, the player controls not one but a number of avatars. From the network's point of view, only the goal of an avatar has to be transported to the other players. All moves or actions can then be computed locally because they all base on the same game logic.

Indirectly controlled games can cope with much higher delay than directly controlled games and the user does not notice delays of several hundred milliseconds. Examples of indirectly controlled games are real-time strategy games such as the Warcraft or Command and Conquer series or role playing games like Diablo.

However, there are still cases where some players do notice higher delays (≈ 500 ms). These player micromanage their avatars by giving only short term goals e.g. influencing the exact way on which the avatar moves to its destination. This is usually done because the player thinks that he knows a better way of doing things than the game logic. In cases where the latency between the players is high, the player's perception of the responsiveness of the game is low during micromanagement of his avatars. Nevertheless, in [17] the authors have looked at the effect of latency in Warcraft III and show that players can adapt to these situations by giving up micromanagement and defining more long-term goals for their avatars. Although players may be unfamiliar with this playing style, tests have shown that the game remains playable and enjoyable. Furthermore, [17] also shows that changing the game style has little or no effect on the game score. It is also a generally accepted fact from psychology and organization theory that the maximum number of characters that a player can control at one time is limited[18, 19]. In [3], we also discuss Warcraft III and found that the skill of the player was the most important factor of the game outcome and that latency and jitter played a negligible role. We conclude that games with indirectly controlled avatars have less demands on the network as games from the previous type. This is

mainly due to the fact that avatars move and behave on their own which allows for an increased delay.

2.3.3. Round-based games

Round-based games are a classical game style which is well known from card and board games. Here, players make their moves mostly in a round-robin fashion. Usually, each player is allowed only a certain number of actions per round. In round-based games the move of a single player may be also limited by time either per round or as a sum for the duration of a single game. Examples for round-based games are chess or poker. For computer games, Civilization is a popular representative of this category.

Round-based games impose comparatively small requirements on the network. They either do not enforce time limits at all or the game imposes time constraints that are in the order of several minutes or more and therefore several orders of magnitudes higher than the delay usually found in computer networks. Games like chess, for example, are also played by mail where moves can take up to several weeks or more. Also web- or flash-based multi-player games generally work in this fashion.

Round-based games can be differentiated from the previous two categories in that they do not allow a seamless and continuous experience of the game by the player. The game is interrupted after each round. Because game data does not need to be sent in real-time, deficiencies or errors in the network can easily be faced with retransmissions of game data on the transport layer. Round-based games usually choose TCP as transport protocol.

2.3.4. Hybrid games

The fourth and last category of multi-player games are hybrid games. This game type combines ideas and elements from the previous three categories. One example for hybrid games is Earth & Beyond, a massively multi-player role playing online game (MMORPG). Here, players directly control their space ship or avatar when moving through the game world space. You will see other players passing by or flying next to you giving you the feeling of a synchronous real-time game. With Earth & Beyond as with all other directly controlled games, all local movements are computed and displayed locally. Thus, unless you interact with other players, that illusion of perfect synchronisation remains true because the player just experiences smooth and consistent behaviour from other avatars even though such actions may be the result of a latency compensation algorithm. During interactions between players, e.g. fights

	Typical Requirements			
Game type	**Bandwidth**	**Delay**	**Jitter**	**Packet loss**
directly controlled	low	high	high / medium	medium
indirectly controlled	low	medium	medium / low	medium
round-based	low	low / no req.	no req.	N/A
hybrid	low	low	low	N/A

Table 2.1.: Network requirements of multi-player games during the game

or trades, games such as Earth & Beyond often resort to a round-based mechanism to ensure a fair game between the player even if one player is impaired with a high latency connection. Hybrid games often use TCP because they are able to hide delay effectively in game and may profit from a reliable connection.

Table 2.1 shows all game types and their respective network categories. We have chosen to present qualitative expressions only to reflect the requirements in comparison between different game types and to other latency-sensitive applications. Maybe aside from an initial setup phase where data is synced between the players, all game types have in common that they use comparatively little bandwidth. As such, multi-player games cover only some parts of latency-sensitive applications. But as we will see later on, even applications with low bandwidth requirements can provide a challenge for the wireless data networks of today. Round-based and hybrid games have the least network demand and because they often use TCP as transport protocol, packet loss is generally of small importance for these games. More information about network requirements for multi-player games as well as quantitative results and results from player interviews can be found in related work, e.g. [20, 17, 2, 14, 21, 3, 22, 23].

For this thesis and for the purpose of supporting latency-sensitive applications in mobile ad-hoc networks, we will focus on directly controlled multi-player games because they have the strongest network requirements. This is understandable because players with direct control over their avatars expect an instantaneous response to their actions. While we put our focus on this single game type, we believe that our work will be applicable to all kinds of multi-player games. If we are able to fulfill the strongest requirements from Table 2.1 then we can also support all three other game types. We have chosen the first-person shooter CounterStrike as our example application for this work. CounterStrike is a popular game that even though that it was introduced in 2001 has still a strong community even today. With CounterStrike, two teams of soldiers compete against each other. The goal is either to complete a mission goal or to kill all members of the opposite team. Today, CounterStrike is one of the most famous

mult-player games with professional players and international championships. From a network point of view, CounterStrike has typical network requirements for a directly-controlled game and it's network behaviour is well researched. An analysis of the network traffic as well as network model of CounterStrike traffic can be found in [8]. In [3], we have presented a survey on the player's idea on network requirements as well as a practical evaluation of several multi-player games including CounterStrike.

2.4. Mobile Gaming

During the last ten years there has been a continuous transition from stationary desktop PCs towards mobile devices. Due to advancements in technology, laptops nowadays provide suitable performance that is sufficient for most applications. In addition, the newly gained mobility allows to utilise computers in a more convenient and flexible way. In the third quarter of 2007 and for the first time ever, mobile computers have outsold stationary devices in the region of Europe, the Middle East and Africa. This includes private sells as well as devices sold for business purposes. The market research company IDC states that 11.2 million laptops and 10.7 million desktop PCs were sold in the third quarter of 2007. Compared to the previous quarter, desktop PCs sales increased by 2.5 percent while laptops sales went up 48 percent during the same time. The general development towards mobile devices lets users now use their application on their hardware everywhere they go. For video games, this evolution already happened in the late 80's with the invention of Nintendo's GameBoy[24] or Atari's Lynx[25] . However, these early handheld game consoles were designed for single player games only.

Additionally during the same time period wireless data technologies such as WLAN or Bluetooth became popular. Other technologies such as GSM or UMTS offer data communication in addition to mobile telephony. Nowadays, mobile access to data networks such as the Internet is available through a number of access points. In urban environments, a large number of WLAN stations offer Internet access. GSM is commonly available all over the country with UMTS catching up slowly. However, with WLAN only a small fraction of all access points share a central coordination which aggravates roaming. In contrast, GSM/UMTS systems have a more centralized organization and a small number of larger providers which comes from the fact that the frequencies for these technologies are exclusively allocated. Here, however, communication is expensive if a larger amount of data needs to be transferred.

The combination of powerful mobile devices and high-performance wireless data technologies fosters mobile computing where any application can virtually be run every-

Figure 2.3.: Mobile video consoles: Sony Playstation Portable[26] and Nintendo DS[27]

where. For games this means that the evolution which began with the GameBoy can now be continued with mobile multi-player games. Again and similar to the development in the PC area, the video game market is also becoming more focused on mobile devices. According to the BIU (Bundesverband Interaktive Unterhaltungssoftware e.V.) and the GfK (Gesellschaft für Konsumforschung AG), the market share for mobile gaming handhelds went up from 8% in 2004 to 19% in 2006. Examples of mobile game consoles are Sony's Playstation Portable (PSP) or the Nintendo DualScreen (NDS) both of which can be seen in Figure 2.3. In the next section, we will describe them in more detail as well as other developments in the mobile gaming area. Further down, we will discuss three mobile gaming scenarios in which these or similar devices are likely to be used for multi-player gaming.

2.4.1. State of the art

The Playstation Portable is a handheld console which encompasses a large screen in the center and various controls on the left and right side of the console. The left side contains four buttons for up, down, left and right movement. On the right side you can find four action buttons. The user interface design is similar to the buttons found on the game controllers for the stationary Playstation 3 game console. Sony introduced the PSP in December 2004 and sold 41 million units worldwide as of August 2008. Today, more than 500 different titles are available for the Playstation Portable with more than 350 games that support multi-player operation. Here, the PSP communicates with other players via it's IEEE 802.11b-based WLAN adapter. It can be used to connect to

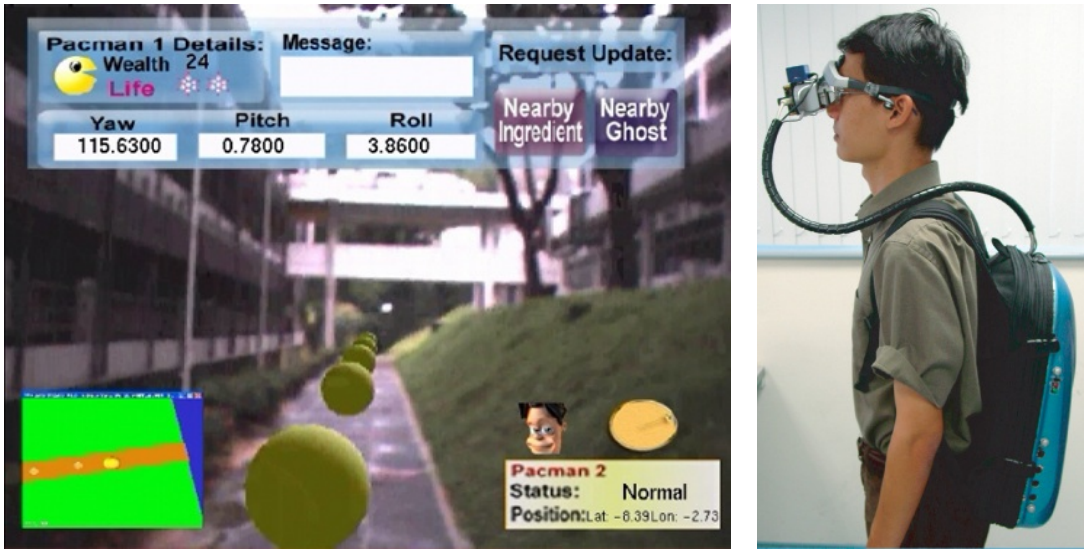
a WLAN access point so that multi-player games can be played through the Internet. It also allows the creation of an ad-hoc wireless network in which 2–16 players can compete against each other without the need for any additional network infrastructure. Some titles for the PSP offer a so called "gamesharing" mode which allows multi-player sessions between two or more players with only one copy of the game. In this mode, the game owner's PSP transmits the game code wirelessly to the other PSPs where it is loaded into RAM. Gameshare versions often support only a reduced feature set of the original game. One reason for this beside the obvious copyright issues is to limit the transfer time of the game code to the other playstations which is done wirelessly at a maximum rate of 11 MBit/s.

The Nintendo Dual Screen handheld console was also introduced in December 2004. As of September 2008 more than 84 million units were sold worldwide[28]. The most prominent feature of the Nintendo DS is that it contains two separate LC displays. The lower LCD is touch-sensitive and can be used either with the included stylus or with the users' fingers. The NDS is backwards compatible with Nintendo's GameBoy Advance. For multi-player games, the Nintendo DS employs a subset of IEEE 802.11b for wireless communication. It supports only data frames with short preambles and communicates at speeds of 1 and 2 MBit/s, possibly with a lower than normal signal strength to save power[29]. Another reason for that behaviour could be that the Nintendo DS broadcasts its frames so that only the basic rates of IEEE 802.11b[30] can be used. The Nintendo DS supports up to 16 concurrent players which can communicate through an ad-hoc network or WLAN access points. As for the PSP, the Nintendo DS allows networked play with only one copy of the game in the network, called "DS Download Play". However, not all NDS games support DS Download Play but require that all players have their own copy of the game inserted into their consoles. Again, some games support only a reduced set of functionality when used in DS Download Play mode. Additionally, DS Download Stations give game stores the opportunity to provide free demo of new games to potential customers. Technically, download stations work in the same way as DS Download Play – DS Download Stations simply contain a couple of Nintendo DS devices with special software cartridges. Currently, there is no freely available information on the technical details of DS Download Play. More on WLAN and game traffic analyses with respect to the Sony PSP and the Nintendo DS can be found in [29]. In addition to player multi-player games, both consoles also allow to play a single-player game with or against the computer.

Besides dedicated consoles, most mobile phones can also be used for entertainment. This includes so called smartphones as well as ordinary mobile phones. In 2003, the finish telecommunication company Nokia developed its Nokia N-Gage, a combination of a mobile phone and a portable video console. For smartphones, games are developed

specifically for their operating system, e.g. Symbian OS, Windows Mobile or iOS. Most other mobile phones can run Java-based games. Because Java games can support lots of mobile phones from different manufacturers, a variety of single or multi-player games exist. Examples of such games are Tetris or Poker. But also MMORPGs can be played on mobile phones, either as a game specifically designed for that platform or in addition to a PC-based game. From the network point of view the variety for multi-player games ranges from games that exchange messages over SMS to games that use their GPRS/UMTS link to communicate with servers in the Internet. In between, there are peer-to-peer multi-player games that communicate directly between two mobile phones via Infrared, Bluetooth or WLAN. In [31] and [32], we introduced a mobile gaming platform designed for the 3GPP IP Multimedia Subsystem (IMS) in UMTS networks. For directly-controlled multi-player games or latency-sensitive applications in general today's GPRS networks are unsuitable because of the high round trip times which often reach 700 ms or above. To a lesser extent, the same is true for UMTS networks. With UMTS, the round trip times of GPRS are roughly cut in half but this is still too high for fast-paced games. Thus, most multi-player games for mobile phones operate in a round-based fashion when using these networks. This restriction however does not affect peer-to-peer multi-player games. Besides common game genres like card or trivia games, mobile phones were among the first device that offered location based games which includes the users' current location as one parameter in the game. The position is either gathered from an integrated GPS chip, from the position of the network cell currently used by the phone, or through delay measurements from different GPRS/UMTS base stations. A list of location-based games can be found in [33] and [34].

Finally, mobile gaming allows for a number of new genres that enables the player to integrate the real world into the game. In related work, the terminology 'pervasive games' can be found for these new game types. Such genres include the aforementioned location-based games and augmented reality games. In [35] and [36], a mobile version of the well known game Pacman called 'Human Pacman' is presented. Players can take the role of Pacman or that of a ghost. Pacman's task is to collect all cookies in the game while the goal of the ghost is to catch Pacman. The cookies are superimposed on Pacman's view of the world as can be seen in Figure 2.4(a). The equipment for a player is shown in Figure 2.4(b). In includes a head-mounted display, a backpack for the required computer hardware and batteries, and several sensors which allows to detect if a ghost has actually catched Pacman. With this prototype, all players communicate with a centralized server through wireless LAN. Another example of a new game type is Real Tournament[37]. The goal of this game is to catch several monsters in a public park. The game is designed so that multiple players are required to catch them in a group effort. Each player is equipped with an iPAQ PDA which shows the



(a) View of Pacman (b) Pacman Hardware

Figure 2.4.: Human Pacman by the Mixed Realtiy Lab, Singapore (from [38])

position of all monsters as well as other players. In contrast to Human Pacman, the monster are not played by any players but are computer generated and exist only on the display of the PDAs. All players move through the park and regularly update their location which is determined by GPS. All Real Tournament clients communicate with a central server by GPRS.

Other trends, like the Nintendo's WII video console follow a different approach. Here, the player holds a game controller in his hand while an infrared camera in the controller detects multiple infrared light sources mounted on top of the display unit (usually a TV set). Images from the camera are then used to determine the exact position of the controller in the room. Further, an accelerometer gives additional sensor feedback. Information from the controller is transmitted back to the game console wirelessly via Bluetooth. The game console itself is stationary and usually connected to a TV set. However, the WII shows the latest example of gaming devices with integrated sensors which allow for the inclusion of real-world information into a game. Sensors such as localization/positioning, acceleration and tilt as well as other data from the environment of a player can complement a game with enhanced reality and immersion. Hence, we see the addition of environmental or personal sensors as an important step in the evolution of computer games in general. Together with the newly achieved freedom of mobility, this development could lead to new game types where players can freely move around and where interaction with the real world is an essential part of the game itself.

2.4.2. Scenarios

Generally, scenarios are important because they define the neighbourhood and the environment of an application. From a given scenario we can deduce not only information about the general condition of the network but also about the number of network users, possible background traffic, mobility and many other parameters. We believe that comprehensive scenarios are essential for a realistic evaluation of any application or algorithm. As an example, it is a fair assumption that rather few people will play multi-player games during business meetings or a lecture class but the majority will enjoy them in their spare time. Also, when at home, it makes more fun to play a sophisticated game on a desktop computer or game console which is attached to a big screen than on rather tiny displays of a mobile device. Therefore, on the following pages, we will introduce three different scenarios which we believe are plausible plots for mobile multi-player gaming. The general idea behind all scenarios is that the player needs to have the opportunity as well as the ability in a particular situation to play a game.

In our first scenario, we narrow down our focus group to pupils or students because we believe that in the beginning new game ideas and technologies will be mainly successful with young people. Therefore, we propose a school yard to be our first scenario. Here, between classes, students come together on the school yard to occasionally play a multi-player game. We believe that among other activities during breaks, multi-player computer games will become a factor in this scenario if they are not already.

In our next two scenarios we focus on the main advantage of mobile gaming which is that you can play them wherever you are. This is especially interesting when you are on the move because you often do not have any network infrastructure to rely on. Hence, we have chosen a train station and a train ride as scenarios for mobile gaming. In both scenarios the player has the opportunity to play a game because he waits for his train to arrive or to reach its destination. On the other hand, when traveling on the train the player has the ability to play the game because he is able to pay the necessary attention to the game.

In all three scenarios, players usually do not have access to any PC or fixed game consoles. Also, while the occasional access to the Internet may exist in any scenario through WLAN access points or mobile networks, availability may be intermittent and access is often not free of charge. While our scenarios do not imply a claim of completeness, we have specifically chosen both indoor and outdoor scenarios which exhibit many different characteristics. While we focus on multi-player games for our scenarios, the scenarios are valid for many other mobile multi-user applications as well.

We define a scenario as the combination of a location together with the environmental

factors and contexts that come with it. We will scatter our mobile devices and users across the scenario to measure the behaviour of our game application algorithms. From a scenario, we collect a variety of parameters, first and foremost, the size and shape of the area. Other parameters include the number of people inside and outside the area of interest, their likely movement pattern and speeds as well as the presence and kind of background traffic. But there are limits to what can be concluded from our scenarios. E.g., the penetration of mobile devices and the percentage of devices which are switched on at the same time cannot be directly determined by looking at our scenarios. For the purpose of this thesis, we will estimate these parameters.

Scenario 1: Schoolyard

For this scenario, we take a look at a medium-sized german grammar school with 600 pupils with an outdoor school yard. Without loss of generality, we define our school yard to be an rectangular area with a size of $200\,\mathrm{m} \times 100\,\mathrm{m}$. We assume a penetration of 5–10 % which results in 30 to 60 devices on the school yard. School breaks usually last between 5 and 30 minutes with 15 or 20 minutes being the typical duration for larger breaks. We therefore, define a standard time of 15 minutes for our scenario.

Figure 2.5 shows an example of a german school yard. It often contains trees and is enclosed by buildings on some sides which may affect wireless communication. While all players could scatter across the school yard, we believe that they will stay close together to enjoy the company of their fellow players. In cases where the game allows one or more teams to compete against each other, we expect a grouping of players either close to their respective team members or as whole group. Besides such clustering of players, we include the occasional remote player where he/she stands or sits alone and focuses on the game. Also, we believe that movement in this scenario will be low when traditional multi-player games are played as to prevent players from walking into things. However, with location-based or augmented reality games that exploit the fact that the user is able to move during the game, we will most likely see lots of movement of all players. Most likely in these cases, players will move on foot. Hence, we propose to use the Gauss-Markov movement algorithm with speeds of up to 4 m/s to simulate pedestrian movement. Despite that, we assume that in our schoolyard scenario traditional multi-player games are used and that players will hold their positions during the game. Regarding background traffic, we assume that we have only small and not significant background traffic on our school yard. In combination with our stationary players, our school yard scenario can be considered close to a best-case scenario for mobile gaming.

Figure 2.5.: A school yard

Scenario 2: Train station

For the second scenario, we take a look at a train station where people walk from and to the train platforms or wait on platforms for their train to arrive. Similar to the first scenario we assume that players do not move during the game as not to walk into things or other people. We also assume a penetration ratio of mobile devices of 5–10%. We use an area of $150 \times 350\,m^2$ for our train station which resembles a medium-sized station. In order to achieve a suitable number of mobile devices, we assume 200 people be present in this scenario which is not unreasonable during rush hour. Because trains run according to a fixed schedule, people usually spend a limited amount of time at the station to get either on or off a train. We consequently set a duration of 15 minutes for this scenario.

The train station is an indoor area where walls can influence or shield wireless transmissions. Furthermore, a train station may have an existing wireless infrastructure network such as a WLAN that allows visitors access to the Internet or is used for internal purposes. Communication from or to this infrastructure may interfere with transmissions from our game if both utilize the same frequency. For data traffic between platforms, communication may be impaired if a train blocks the line of sight between the communication partners. The effect is shown in Figure 2.6 where a WLAN link between two platform was impaired by a train. Finally, electromagnetic noise from the overhead line, the engine of an electric locomotive or other equipment also has a negative effect on wireless communication.

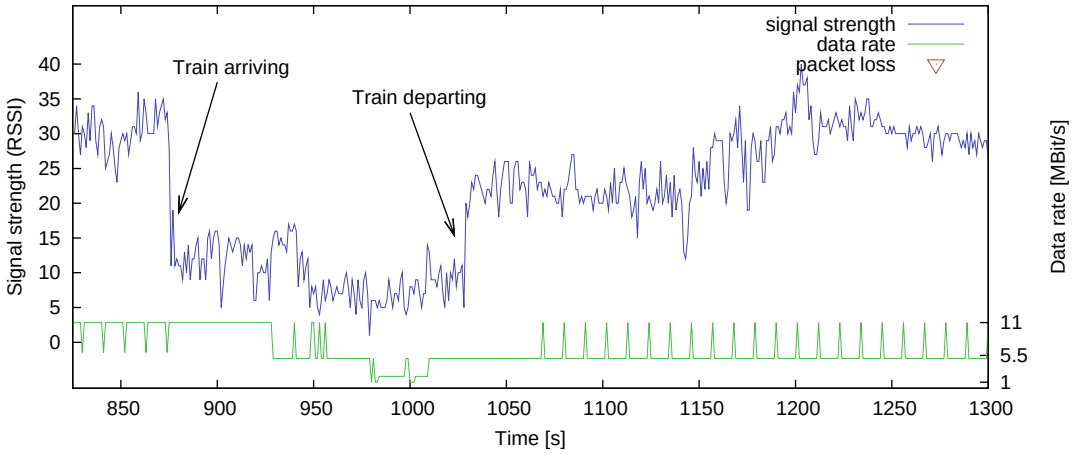


Figure 2.6.: Signal strength and data rate between two train platforms

In contrast to the school yard scenario, at the train station other people are participating in the wireless network as well. There could be people using different applications which again communicate directly over the wireless network or talking to computers in the Internet through existing infrastructure at the station. Some examples of that traffic can be business people sharing an important company document with each other or a visitor just surfing on the Internet. From the point of view of the game, data from these applications is considered background traffic which competes with our game traffic for the use of the wireless channel. Also, in contrast to our players, background nodes are able to freely move around at the train station. This comes from the fact that mobile devices may communicate without any user interaction, e.g. synchronising email, which leaves the possibility for their owners to walk around. We also utilize the same pedestrian mobility model that we used in the schoolyard scenario.

Scenario 3: Train ride

For the train scenario, we use a common long-range InterCity train like the ICE 3 built by Siemens/Bombardier/Adtranz. This particular train type is often used with eight coaches which are each 3 m wide and have a combined length of 175 m[39]. This train has a standard capacity of 431 seats. Assuming that two thirds of these seats are occupied, which is a reasonable assumption for popular connections, our ICE 3 has 287 people on board. With our standard penetration ratio of 5–10 %, we have between 15 and 30 mobile devices operating on our train or approx. two to four devices per coach. For our train ride we use a scenario duration of 60 minutes. The train scenario

Figure 2.7.: A train platform with an InterCityExpress

is characterized as an indoor scenario. Inside a coach wireless communication may be influenced by the interior design of the coach and other passengers. Wireless communication in a straight line-of-sight is possible only in exceptional cases. Between two coaches, the wireless signal is attenuated by the metal shell of the coach itself. In our WLAN measurements, we found that inter-coach communication to be problematic and significantly impaired with today's hardware as can be seen in Figure 2.8.

In this scenario, two different situations should be distinguished. First of all, during the train ride, people usually sit at their seats and do not move often. The second situation is a stop at a train station when people get on or off the train and in the following minutes while they make their way through the coaches. So again, players are not expected to move during their game and the mobility of background nodes is also low as long as the train is on the move. Again, people travel on foot and because the environment is narrow we assume a reduced maximum speed of $2\,{}^{\mathrm{m}}/_{\mathrm{s}}$. During the ride, user mobility is generally restricted forwards and backwards along the center line of the train.

In this scenario, we expect a number of other nodes that operate concurrently with our multi-player game. We expect file sharing between two nodes and Internet access to be among the top applications during a train ride. In Germany and as of June 2011, Deutsche Bahn has equipped 69 ICE trains with wireless Internet access which is offered in cooperation with T-Mobile on four different routes across Germany[40]. We

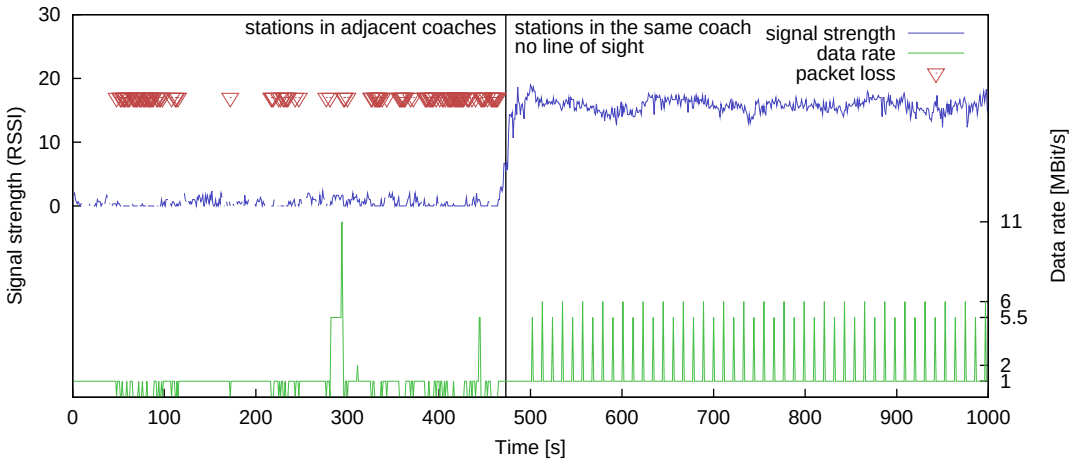


Figure 2.8.: WLAN measurement results inside a train

Figure 2.9.: First classs coach of an InterCityExpress (ICE2)

Scenario	Node density	Mobility	Background traffic	Obstacles
Schoolyard	o	o	–	o
Train station	-	+	+	+
Train ride	+	–	+	++

Table 2.2.: Scenario comparison

will use between 7 and 15 nodes for background traffic which represents between 25 % and 75 % of all mobile devices in the train. As the train travels through cities and the countryside rapidly, other wireless stations outside the train may influence wireless transmissions inside the train or even come in contact with devices in the train. Depending on the direction of travel, the relative speed between such two devices may be up to $100\,{}^{\mathrm{m}}/_{\mathrm{s}}$.

2.5. Evaluation criterias

Table 2.2 shows the different characteristics of all three scenarios in comparison. We can see that each scenario has its own challenge. For the node density, we calculated the number mobile devices present in relation to the area size in the specific scenario. We designed our scenario for multi-player games so that they fit the requirements of games available either on existing game consoles or on PCs today. This action specifically does not take any special requirements for new mobile game genres into account. Especially, we decided that players do not move during the game as interaction with the environment is not required during the game. We have specifically chosen this path in order to limit the amount of different mobile gaming scenarios and because the detailed network requirements for location-based and augmented reality games are not yet clear. Table 2.3 shows a more detailed list of the parameters for our three scenarios.

As application for this thesis we have chosen the CounterStrike, a directly-controlled multi-player game and a fast-paced first-person shooter that has high network requirements. In [3] we have shown that a round-trip delay of 150 ms is acceptable for CounterStrike and that the game can cope well with delay variation of a similar extent. [8] showed that CounterStrike sends and receives 20 packets per second thus using a total bandwidth of 4 kbit/s per client. There are no specific requirements regarding the maximum tolerable packet loss for CounterStrike in related work. However, similar games like Unreal Tournament or Quake 3 allow for packet loss between 3–5 % before the gameplay is noticeable impaired for the user[13][2]. Therefore, we also use an upper limit of 5 % for packet loss with CounterStrike.

	Schoolyard	Train Station	Train
Size of area ($w \times l$)	$200\,\mathrm{m} \times 100\,\mathrm{m}$	$150\,\mathrm{m} \times 350\,\mathrm{m}$	$3\,\mathrm{m} \times 175\,\mathrm{m}$
Main obstacles	Trees Buildings	Walls Trains	Other people Carbodies
People (total)	600	200	287
Device penetration ratio		5 – 10 %	
Nodes running			
game application	30 – 60	5 – 10	7 – 15
other apps (background)	none	5 – 10	7 – 15
Mobility (background nodes)			
Model	N/A	Gauß-Markov	Gauß-Markov
Node speed	N/A	$0 - 4\,\mathrm{m/s}$	$0 - 2\,\mathrm{m/s}$
Other movement	N/A	N/A	up to $100\,\mathrm{m/s}$
Time	15 min.	15 min.	60 min.

Table 2.3.: Parameters from the scenarios

In [3] we have also shown that the user's impression on the network parameter required for a multi-player game can differ from its real requirements. Still, we believe that achieving the aforementioned network characteristics in a wireless networks is sufficient to support any mobile multi-player games. Later, in Chapter 6, we create mobile gaming scenarios and evaluate the performance of a multi-player game based on information from this chapter.

2.6. Problem statement

In the remainder of this thesis, we focus on current multi-player games as an example of latency-sensitive and interactive applications. Ensuring the timely delivery of data for these applications or a certain application-specific quality of service for data communication can sometimes be difficult. In the Internet, this problem is often solved by overprovisioning. If more than enough bandwidth from source to destination is available at all times, no bottlenecks occur. In wireless networks, the bandwidth available to applications is several orders of magnitude lower than in the Internet. Here, if too many users access the wireless network concurrently or some applications have high bandwidth demands, congestion can happen easily. In a wired network, this problem can further be alleviated if network links are used by a single computer only, e.g. in

a switched network. Wireless networks are a shared medium by principle and every sender influences other wireless stations in its vicinity. For these reasons, applications that work well in the Internet do not necessarily work well in wireless networks. This is especially true for applications that have a certain demand on the quality of the network, such as multi-player games or latency-sensitive applications in general. Finally, user mobility introduces additional problems that do not exist in wired communication.

Wireless networks often work by using a centralized base station which controls and/or coordinates access to the network. All traffic between two wireless stations as well as all data to the wired network always runs through these base stations. While a centralized system can help against some problems in wireless networks, they are not available everywhere. Moreover, maintaining a wireless infrastructure is expensive, thus access to those networks is often not free of charge. Wireless ad-hoc networks that are used by today's mobile game consoles communicate between mobile devices and can be created on demand. Still, they are not able to span larger distances because they need to communicate directly.

In this theses we aim at enabling multi-player gaming in mobile networks without the need for any wireless infrastructure. As discussed in this chapter, we hereby focus on existing mult-player games that are used in the Internet today. To achieve that goal, we aim at creating a flexible server-based architecture for mobile gaming which includes redundancy, supports mobility and is scalable. We further look into methods how wireless communication for latency-sensitive applications can be enhanced without penalizing other traffic at the same time.

2.7. Chapter Summary

In this chapter we discussed different latency-sensitive applications in which two or more users interact with each other. A popular example of this type of application are multi-player computer games which we will use as an example for the remainder of this thesis. They have a variety of network requirements ranging from modest round-based games to fast-paced games where the user directly controls his avatar. Through our own work and from related work, we conclude that such an application requires a round-trip delay of less than 150 ms and a maximum packet loss of 5 %.

We then discussed the state of the art in mobile gaming where games are played on mobile gaming consoles like the Nintendo DS or nowadays on an increasing number of smartphones. Existing wireless technologies like WLAN or Bluetooth are used to communicate between devices in a peer to peer fashion. Mobile users that play with each other therefore need to stay in close proximity to each other to allow a direct

communication between devices. We believe that today mobile gaming dominantly takes place in specific scenarios that are characterised by the lack of a larger, more advanced gaming devices and that give the user the opportunity to play a game. We presented three different scenarios namely the schoolyard, the train station and the train ride scenario

Finally, we briefly present the problems that latency-sensitive applications on mobile devices have with their specific network requirements in wireless networks. We conclude this chapter by presenting our goal to create and select the necessary architecture and technology to support existing multi-player games in realistic mobile scenarios using a wireless network.

3. Technologies & Architectures

Providing a suitable network for multi-player games in particular or latency-sensitive applications in general is a challenging task. Luckily, with advancement in data transmission technologies as well as increasing capacities in routers and bridges, the Internet of today offers high bandwidth and low latency across countries and to a certain extent continents to suit their needs. In mobile ad-hoc networks, such resources are still scarce and limited. Furthermore, radio networks have their unique characteristics and problems that have to be taken into account. New challenges such as mobility ask for a new kind of architecture that relies on a distributed rather than a centralized approach.

In the following chapter we will introduce wireless technologies and concepts as well as communication architectures that we use for our work. We will present existing approaches, discuss their problems and point out solutions where possible. We have divided this chapter into two major parts. Firstly, we provide background information on wireless radio technologies and mobile networks. We start by giving a brief overview of the wireless local area network (WLAN) protocol family that we will use in our evaluation later on. We put special emphasis on problems that latency-sensitive applications encounter in these networks. Furthermore, we take a brief look at mobile ad-hoc networks (MANETs). We then move on to communication architectures. We present existing architectures from the Internet as well as related work for new approaches in mobile ad-hoc networks. We also examine architectures for multi-player games. Finally, we present our zone server architecture as a scalable approach for multi-player games in mobile ad-hoc networks that provides redundancy, supports mobility and allows data aggregation to save bandwidth.

3.1. IEEE 802.11 Wireless Local Area Networks

To support mobile clients, a radio communication protocol is required that deals with the special characteristics of the wireless channel. One well-known example of such protocols is Wireless LAN according to the IEEE 802.11 standard. In this thesis we will use the terms 'Wireless LAN' and 'WLAN' as synonmys for this IEEE standard.

WLAN can be used to build mobile ad-hoc networks with high data rates and corresponding small transmission times and therefore low latencies. We will introduce Wireless LAN and mobile ad-hoc network in a bottom up approach, starting with the radio communication protocol before we move up to a more general view of mobile ad-hoc networks. In this section, we begin by giving a brief introduction and overview of the Wireless LAN family. We then move on to specific radio communication issues that result in a different network behaviour when compared to common wire-bound networks. We hereby put our focus on latency and fairness which can have a significant impact on latency-sensitive applications. We conclude this section with a theoretical analysis of the capacity of a wireless network using a multi-player game.

3.1.1. Introduction

The IEEE 802.11 Wireless Local Area Network (Wireless LAN or WLAN) standards family was originally created to complement or replace wired local area networks. It was specifically designed to work seamlessly with existing wired IEEE 802.x technologies such as e.g. Ethernet[41] or Token Ring[42]. All mentioned technologies share the same higher layers starting from logical link control (LLC) upwards. Details that are relevant to a specific transmission technology vary only in the physical layer and the medium access control sub-layer. Thus, data transfer between different technologies can be done with a simple bridge device rather than a more complex gateway. WLAN uses a method called Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) instead of collision detection as in wired Ethernet. This is due to the nature of the radio medium which does not allow to detect collisions reliably.

In 1997, the Institute of Electrical and Electronics Engineers (IEEE) approved the first WLAN standard IEEE 802.11[43]. It allows transmission rates of one and two MBit/s and works in the 2.4 GHZ frequency band which is license-free for industrial, scientific and medical purposes (ISM radio band). 802.11 describes three different physical layers: Distributed Sequence Spread Spectrum (DSSS), Frequency Hopping Spread Spectrum (FHSS), and infra-red. In the course of further development of WLAN, DSSS has become the most important physical layer for WLAN. Newer WLAN enhancements do not consider FHSS or infrared transmission modes any longer. To increase resistance to interference, DSSS spreads the wireless signal across frequencies using an 11-chip Barker code.

Two years later in 1999 two additional standards were published. IEEE 802.11b extends the original standard with two additional high-speed data rates of 5.5 and 11 MBit/s while remaining downwards compatible. 802.11b also proposes a shorter frame format that reduces the initial synchronization in the preamble by more than

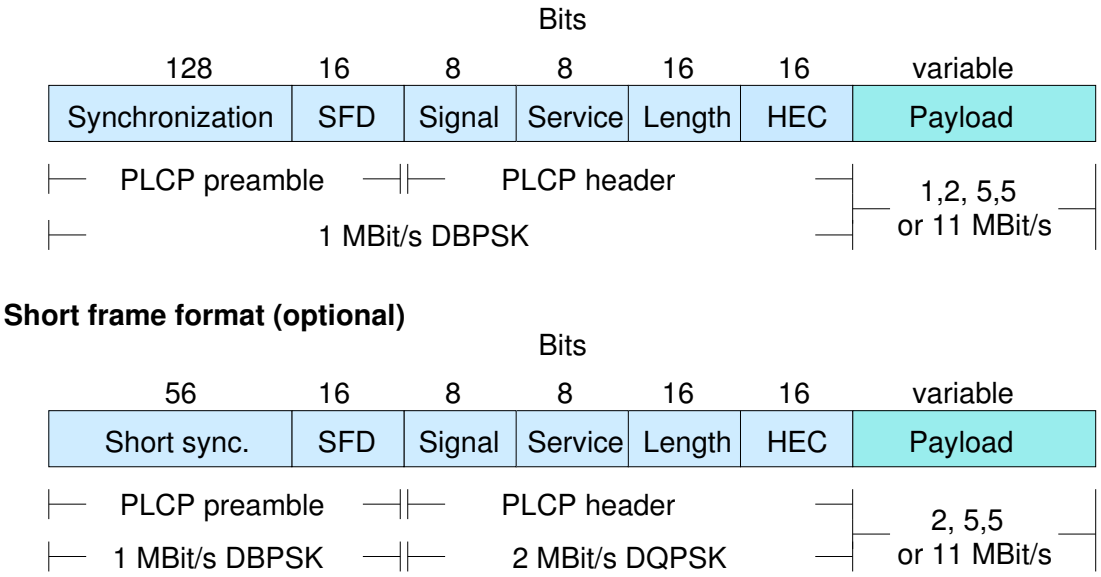


Figure 3.1.: IEEE 802.11b frame format with short preamble

50 %. Figure 3.1 shows both frame formats and includes both preamble and header of the Physical Layer Convergence Protocol (PLCP) as well as data. Preamble and header are transmitted at comparatively low data rates to ensure carrier sensing as well as successful reception at the receiving side. The second standard published in 1999 was IEEE 802.11a which uses orthogonal frequency division multiplexing (OFDM) and allows data rates up to 54 MBit/s. This standard operates in the 5 GHz band and did not achieve widespread popularity because this frequency band is regulated in most countries. Consequently, an OFDM-based WLAN standard for the 2,4 GHz ISM band was developed in 2003. This standard is called IEEE 802.11g and is basically an adaption of 802.11a for the lower frequency band with only minor changes. It inherits the same data rates and properties from its predecessor as well as a slightly different frame format. However, additional features were specified for 802.11g to allow coexistence with existing WLAN implementations based on 802.11b. Today, most WLAN devices support either the 802.11b/g or the 802.11a/b/g standards.

In recent years, additional WLAN variants have been developed or are in the process of being standardized. Most notably, the new multiple-in multiple-out (MIMO) WLAN specification IEEE 802.11n (draft standard) offers even higher data rates and is believed to become the successor for 802.11a/b/g. Other WLAN developments target specific problems such as long-range point-to-point links or individual scenarios like inter-vehicular communication. Table 3.1 gives an overview of current and future

IEEE Standard	Year published	ISM band	Frequency	Gross data rate
802.11	1997	yes	2400 – 2485 MHz	up to 2 MBit/s
802.11a	1999	no	5150 – 5350 MHz 5470 – 5725 MHz	up to 54 MBit/s
802.11b	1999	yes	2400 – 2485 MHz	up to 11 MBit/s
802.11g	2003	yes	2400 – 2485 MHz	up to 54 MBit/s
802.11n	2009	yes	2400 – 2485 MHz	up to 600 MBit/s
802.11p	2010	no	5850 – 5925 MHz	up to 27 MBit/s (est.)
802.11y	2008	no	3650 – 3700 MHz	up to 54 MBit/s

Table 3.1.: WLAN transmission standards

WLAN standards.

WLAN uses a number of different waiting times between subsequent transmissions so called inter-frame spaces (IFSs). They are used as guard intervals and to prioritise channel access for different kinds of transmissions. Two examples of inter-frame spaces are the distributed inter-frame space (DIFS) and the short inter-frame space (SIFS). DIFS is the time that a sender has to wait after any data transmission before it can start competing with other channels over channel access. It thus defines the minimum waiting period between the transmissions of two different stations. For unicast data, WLAN uses acknowledgments that provide feedback to the sender that the data packet has been received successfully. Such acknowledgments are expedient because interference is quite common on radio channels. In order to be able to give that feedback in a timely manner after the packet has been received, acknowledgements have a higher priority than ordinary data packets. This preference is expressed by using a shorter waiting period (SIFS) which ensures that acknowledgements always directly follow their corresponding data packets. Other prospective senders would have to wait the longer DIFS period which eventually results in a back-off because the medium is occupied by the acknowledgment. SIFS and DIFS are not the only inter-frame spaces defined in the various WLAN standards although their counterparts work accordingly. WLAN introduces an inter-frame space between all transmissions so smaller data packets have more protocol overhead than larger ones.

Most WLAN networks use contention-based medium access using a Distributed Coordination Function (DCF) that treats all stations and their data in the same way. Nevertheless, data from one application, e.g. a video streaming application that the user is watching, may be more important with respect to bandwidth and timely delivery than

data from an backup application.To allow differentiation and prioritization of traffic, two standard methods are defined. First of all, the point coordination function (PCF) allows an access point, the point coordinator, to arbitrate the medium. PCF uses a superframe comprising a contention-free period and a contention-based one. During the contention-free period, the point coordinator polls data from all stations similar to a TDMA scheme thereby guaranteeing a suitable service schedule. PCF is part of the original IEEE 802.11 standard, but its implementation is optional. It has not seen any relevant distribution and is hardly implemented in any commercial WLAN product today.

In November 2005, the IEEE published the 802.11e standard[44] which amends the MAC with Quality of Service (QoS) enhancements. It introduces a third coordination function: The Hybrid Coordination Function (HCF). Similarly to existing coordination function, HCF comprises a contention-based channel access (Enhanced Distributed Channel Access or EDCA) and a controlled channel access (HCF Controlled Channel Access or HCCA) variant. The latter function uses a hybrid coordinator to arbitrate channel access and generally resembles PCF. Again like PCF, it is also defined as optional. On the other hand, EDCA provides distributed channel arbitration with traffic prioritization in eight categories[45]. Data can be classified into eight different traffic categories which are specifically designed to match user priorities defined in IEEE 802.1d[45]. Every two traffic categories are dedicated to specific purposes which are in ascending order of priority: Background, best effort, audio, and video traffic. Channel access prioritization is controlled in a distributed fashion by using shorter or longer inter-frame spaces (IFS) between transmissions. EDCA offers no additional QoS features beyond traffic prioritisiation. More information on WLAN and its standards can also be found in [46] and [47].

Using WLAN for latency-sensitive applications has several implications because of the available bandwidth and how the medium is accessed. During the following sections we will take a look at the implications of using WLAN for latency-sensitive applications. We will especially examine the impact that different traffic patterns have on fairness between stations in WLAN, look at latency and provide a capacity estimate for latency-sensitive applications.

3.1.2. Fairness

To provide for a distributed and preferably collision-free medium access, 802.11 DCF uses an algorithm that defines how stations compete for the medium during contention periods. This slotted binary back-off algorithm also controls a fair access to the medium. Before a station can send, it senses the channel to assess whether it is

free or busy. After the medium has been free for at least a period of DIFS, the station randomly selects a number of slots to wait before transmitting within the contention window. This randomness reduces the number of collisions if multiple stations have data to send. The station with the smallest random slot number wins the contention phase hence send its data first. All other stations will then back off as the medium is no longer free while retaining their current number of slots to wait for the next round. This approach increases the chance of stations that had to back off to be able to win access to the channel at the next opportunity therefore creating fairness between stations.

In the Internet data is often transmitted with packets close to the Ethernet Maximum Transfer Unit (MTU) of 1,500 bytes. Another common packet size is smaller than 100 bytes which is used for protocol management such as TCP acknowledgements or requests for data such as DNS[48, 49] or HTTP[50] requests. Latency-sensitive applications often use small data packets to reduce buffering delay and increase responsiveness for the user. Table 3.2 shows average packet sizes and data rates for different applications. The numbers shown here are for wired Internet traffic and may not be exactly be the same as for future applications in wireless LANs and/or mobile ad-hoc networks. However, we believe them to be applicable as packet size distribution in the Internet did not change significantly over the last 10 years[51, 52].

Fairness achieved by WLAN's back-off algorithm is solely based on the number of MAC frames a station can send. If simililar traffic patterns are assumed, this means that all stations can send an equal number of frames if measured over a longer time. WLAN fairness methods however ignore differences in frame sizes or transmission rates that occur in real-world applications. Thus, the binary back-off algorithm creates multiple drawbacks for latency-sensitive applications which we will briefly discuss in the following paragraphs.

Latency-sensitive applications have to sent more and smaller packets when compared to other applications to reduce delay and increase responsiveness. Thus, they use the medium inefficiently because of a larger protocol overhead. It is a common fact that because the per-packet overhead is constant, the most efficient use of a medium is with MTU-sized packets. However, if we ignore the overhead for a moment and take only the payload into account, wireless LAN still favours larger packets. This is because WLAN uses a contention system that provides fairness on a per-node level. Basically this means that in the long term, it aims at balancing the number of packets that each individual stations can sent. Thus, if we take a standard TCP connection consisting of 1,500 byte data and 52 byte TCP acknowledgment packets and compare it with multi-player game traffic of Counter Strike (see Table 3.2) in a wireless LAN, we can calculate that TCP's throughput is 7.7 times greater. However, this initial result is biased because it ignores differences in protocol overhead.

Application	Common packet size(s)	Direction	Characteristic / Rate
Internet [51]	$<$ 100 bytes	n/a	$\approx$ 50 % of all pkts
(various appl.)	$>$ 1,300 bytes	n/a	$\approx$ 40 % of all pkts
Counterstrike[8, 53]	avg. 82 bytes	client $\rightarrow$ server	24 packets/s
	avg. 127 bytes	server $\rightarrow$ client	18 packets/s
Quake III[54]	82 - 166 bytes	client $\rightarrow$ server	20-90 packets/s
	60 - 120 bytes	server $\rightarrow$ client	20 packets/s
VoIP G.711[55, 56]	160 bytes	bidirectional	2$\times$50 packets/s
VoIP G.729[57, 56]	10 bytes	bidirectional	2$\times$10 packets/s

Table 3.2.: Common packet sizes and traffic characteristics for applications

To better understand the differences between TCP and Counter Strike game as well as VoIP traffic, we have developed a theoretical WLAN simulation model that calculates how long the medium is occupied by a station. It calculates the transmission time for every single frame and includes details such as transmission rates, inter-frame spaces and the contention period. Latter is determined by calculating the average waiting time in the contention period based on the minimum and maximum values of the congestion window. Therefore and for the purpose of this analysis we defined the time period for which the medium is occupied by a transmission as follows:

$$t_{\text{frame}} = t_{\text{backoff}} + t_{\text{data}} + t_{\text{ack}}$$

with

$$t_{\text{backoff}} = t_{\text{DIFS}} + (CW_{\max} - CW_{\min}) * t_{\text{slot}}$$

$$t_{\text{data}} = t_{\text{phy}_{\text{head}}} + t_{\text{mac}_{\text{head}}} + \frac{\text{payload}}{\text{payload data rate}} + t_{\text{padding}} + t_{\text{phy}_{\text{tail}}} + t_{\text{mac}_{\text{tail}}}$$

$$t_{\text{ack}} = t_{\text{SIFS}} + t_{phy_{head}} + \frac{802.11 \text{ acknowledgement}}{\text{ack data rate}} + t_{phy_{tail}}$$

In this calculation, the padding time and the time for the physical tail are zero except for 802.11g data rates. With 802.11g padding may become necessary because data is transmitted not in bits but in symbols which have a fixed bit length for each transmission rate. Our model provides accurate medium occupancy times for WLAN transmissions except for the calculation of the back-off time. The exact duration of

IEEE 802.11b

Transmission Rate	1 Mbit/s	2 MBit/s	5.5 MBit/s	11 MBit/s
CounterStrike game traffic	20.9 %	24.5 %	32.1 %	37.9 %
TCP traffic	79.1 %	75.5 %	67.9 %	62.1 %
Game vs. TCP	1 : 3.8	1 : 3.1	1 : 2.1	1 : 1.6
G.711 VoIP traffic	24.5 %	27.5 %	34.1 %	39.1 %
TCP traffic	75.5 %	72.5 %	65.9 %	60.9 %
VoIP vs. TCP ratio	1 : 3.1	1 : 2.6	1 : 1.9	1 : 1.6

IEEE 802.11g (selected transmission rates)

Transmission Rate	6 Mbit/s	18 Mbit/s	36 Mbit/s	54 Mbit/s
CounterStrike game traffic	25.0 %	33.8 %	39.3 %	42.1 %
TCP traffic	75.0 %	66.2 %	60.7 %	57.9 %
Game vs. TCP ratio	1 : 3.0	1 : 2.0	1 : 1.5	1 : 1.4
G.711 VoIP traffic	28.0 %	35.4 %	40.2 %	42.6 %
TCP traffic	72.0 %	64.6 %	59.8 %	57.4 %
VoIP vs. TCP ratio	1 : 2.6	1 : 1.8	1 : 1.5	1 : 1.3

Table 3.3.: Medium occupancy ratios for competing stations on an unimpaired wireless LAN

the contention period is difficult to calculate because it depends on whether there was an immediate previous transmission or collision in the network. Hence, we use a simple approach to estimate the duration of the contention window which assumes no collisions and the existence of a prior successful transmission which should be close enough for a wireless network with medium load.

We use our medium occupancy time model to compare the times of two station pairs running different applications. The first station pair either uses the multi-player game CounterStrike or VoIP traffic according to the G.711 encoding that are both described in Table 3.2. The station pair that simulate CounterStrike traffic uses client-server as well as server-client traffic characteristics. The VoIP station pair uses the same kind of traffic for both directions. The second station pair creates a TCP stream with 1500 bytes segments with TCP acknowledgements in the opposite direction. We calculate the time it takes to send and receive 10,000 data frames with different data rates for each station pair. Because of WLAN's per station fairness, we can assume that all data frames are distributed fairly between all stations. However, due to overhead from the lower layers and the differences in data payload, the medium occupancy ratio are not evenly distributed. Table 3.3 shows the result of our calculation. We can see that the TCP stream occupies a larger amount of medium utilisation time which can be up to 3.8 times higher than game traffic. This ratio decreases with higher data rates which reduces the impact of overhead of lower layers. Because 802.11b and 802.11g use different transmission schemes (DSSS vs. OFDM), the results cannot be compared directly between both technologies. We can see from our analysis that per-frame fairness of WLAN's back-off algorithm neither provides an equal share of throughput nor medium access time to latency-sensitive applications. Although, this difference is alleviated by higher transmission rates. For latency-sensitive applications like multi-player games or VoIP this means that stations running applications with other traffic characteristics are able to use more network resources than their equal share.

Another point is that wireless LANs cannot guarantee any time-limited transmission for latency-sensitive applications due to the nature of a contention-based access scheme. Therefore, transmitting larger packets with small transmission rates can present additional problems for latency-sensitive applications because they occupy the medium for a disproportionate long time. As WLAN defines no explicit upper time limit besides the maximum frame size for how long a station can transmit after it has won the contention phase, other stations have to wait until the end of the current transmission before they have the chance to compete for channel access. For example, a station transmitting 1,500 bytes at 1 MBit/s takes 12,8 ms from the time it has won the contention period until the channel can be used again. Keeping in mind that WLAN

provides fairness between the number of transmissions of individual stations and the fact that collisions may occur, noticeable delays for transmission from a specific station may occur even with a small total number of competing stations. This problem has been addressed by WLAN's QoS extension[44] which introduces transmission opportunities (TXOPs) for high priority data. With TXOPs, stations can occupy the channel for a specific amount of time which allows them to send either one large packet or several smaller ones. Packets with low priority on the other hand retain their original right to send a single frame. But because IEEE 802.11e increases the chances that a station with high priority data wins the contention phase, the problem is still alleviated. While WLAN's QoS extensions have been published in 2005, still only a few drivers and fewer applications make use of IEEE 802.11e today. Besides contention frame loss is also a contributor to end-to-end delay and delay variation for acknowledged unicast transmissions.

Related work on Fairness

Fairness in 802.11-based WLANs has been widely discussed in related work. In the following, we introduce and discuss publications from selected areas in that field. A number of publications study the effect of unfairness on TCP. Pilosof et al. [58] discuss an unfairness between TCP upstream and downstream flows in IEEE 802.11 networks. Because WLAN provides per-station fairness instead of per-flow fairness, TCP downloads are significantly impaired compared to uploads because the access point counts as a single and equal station in the network but has to serve multiple wireless clients. Pilosof et al. show that throughput ratios of eight or higher between up- and downlink are not uncommon and that the buffer size at the base station plays an important role in the fairness of competing TCP flows. They propose a manipulation of the advertised window size at the base station to alleviate the problem. Blefari-Melazzi et al. [59] proposed a solution based on static or adaptive rate-control mechanisms. They evaluate their approach by measurements in an experimental ad-hoc test-bed. Leith et al. [60] discussed the same problem but in 802.11e networks that provide different QoS queues. They show that the unfairness can be solved by allowing TCP acknowledgments unrestricted access to the channel. This enables TCP to regulate the amount of data and provides fairness between flows. Seyedzadegan et al. [61] reviewed several approaches that provide TCP fairness in WLANs. They differentiate existing research in per-station and per-flow schemes. Thereby, approaches like e.g. DCA[62] or fair[63] modify the medium access layer while DualQ[64] or DATC[65] are based on different queuing and rate control strategies on each station. They conclude that most research uses a one-flow-per-station paradigm which is not realistic.

Another approach in related work is to deal with fairness issues on the MAC layer. Razafindralambo and Valois[66] analysed four different back-off algorithms including WLAN's binary exponential back-off algorithm in a multi-hop ad-hoc scenario and conclude that they are neither efficient nor fair. Vaidya et al. [67] propose the distributed fair scheduling protocol (DFS) which is an extension to the existing 802.11 distributed coordination function (DCF) and based on SCFQ[68]. SCFQ is a flow-based queuing scheme and considers among other parameters the size of the packet. The main difference between DFS and DCF is how the back-off interval is allocated. The authors show that DFS provides a fairer allocation of bandwidth than DCF.

Other related work takes a look at fairness issues from different points of view. Berger-Sabbatel et al. [69] looked at short-term fairness between two stations sending UDP frames of equal size. They show in theory and by simulation that WLAN generally provides good fairness and that it remains acceptable for an increasing number of hosts. Heusse et al. [70] discussed the influence of stations with low transmission rates on an 802.11b infrastructure network. They show that due to protocol overhead involved, a single station using a transmission rate of 11 MBit/s can achieve a maximum theoretical throughput of approx. 7.7 MBit/s under ideal conditions. The author state that if a network of nodes encounters a single node that transmits at 1 MBit/s, the throughput of all nodes is significantly diminished down to about 1 MBit/s. They conclude from theoretical analysis and experimentation that a host that captures the channel for a longer time because of its low transmission rates penalises hosts with higher transmission rates. Nevertheless, their tests were only performed with the same traffic characteristics for all hosts. Casetti and Chiasserini[71] study VoIP traffic in an 802.11e infrastructure network. Through NS-2 simulation, they show that fairness and throughput problems arise when the number of stations increases and traffic is classified solely based on its type. They propose to include information on the direction of traffic as well. They use the Jain fairness index to show that their approach can achieve a much fairer distribution of traffic.

In summarizing related work, we can conclude that fairness remains an issue with WLAN networks. This is mainly because different scenarios and traffic patterns each require a different kind of fairness such as per-flow fairness or an equal share of medium access time. The problem can be alleviated by employing 802.11e QoS extensions which offer more possibilities to define how the wireless medium can be accessed. However, it does not solve the problem entirely and special considerations have to be taken into account when classifying traffic. Many solutions have been proposed in the area of back-off algorithms, coordination functions, queuing, rate-control and custom transport layer adjustments. However, most work focuses on TCP and the effect of different packet sizes and especially individual transmission rates has not

been widely discussed. From our own observations, we have seen that WLAN's per-node fairness scheme can provide a fair medium access only when stations are sending similar-sized packets at similar transmission speeds. Stations with larger packets or lower transmission speeds can generally occupy the medium for a longer time than their neighbours. For latency-sensitive applications which have to transmit their data within the time bounds of their application, this means that the maximum number of stations that can operate at the same time within the network is generally not fixed but depends on the individual traffic characteristics of the neighbours.

3.1.3. Latency

For interactive applications, latency is at least as important as throughput. In this section, we take a closer look on end-to-end delay in wireless LANs. We describe and discuss the various sources that produce latency and differentiate them into various categories.We already discussed in Chapter 1 that for interactive applications, the end-to-end delay between an action and a reaction is important. In network terms this describes the round-trip time of a communication system. Figure 3.2 illustrates the theoretical minimum round trip time (RTT) in an 802.11b WLAN. Assuming that the channel was occupied before, stations have to wait a mandatory DCF Inter-frame Space (DIFS) waiting period before they are allowed to send data. Then, because the receiver must only wait for a shorter period (SIFS), the transmission of the acknowledgment is prioritized over data traffic from other stations. Without any regard for processing time at the receiver, it sends its response after the obligatory DIFS. As soon as the first station has completely received the packet, it can start processing its content. Therefore, the final ACK is not added to the round trip time. Because this figure shows the lower boundary for RTT, we omitted the initial arbitration period for our data. This boundary for round-trip time in WLANs, however, usually cannot be achieved by real devices. Here, additional latency from retransmissions, congestion or packet loss add to the round-trip delay. Furthermore, station internal delays due to interrupts, packet processing, etc. occur. The minimum round trip time for IEEE 802.11a/g networks is basically identical to Figure 3.2 but for small changes in the composition of the data frame.

Given a transmission speed and data size, the minimum round-trip time for an IEEE 802.11b network can be calculated accordingly to Figure 3.2. Let R_p^{req} and R_p^{resp} denote the number of (re-)transmissions of the p-th request and response packet respectively. E.g. a value of $R_3^{\mathrm{req}} = 1$ means that the initial transmission of the third packet was successful and that no retransmission of that specific frame was necessary. Also, let δ_p^{req} and δ_p^{resp} be the queuing and backoff times before the wireless channel can be

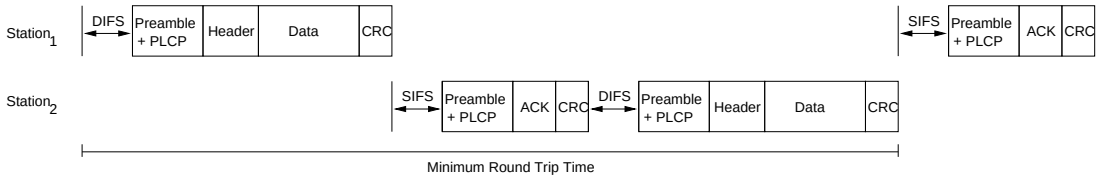


Figure 3.2.: Theoretical minimum round trip time for IEEE 802.11b

successfully accessed and Φ_p the processing time of the p-th packet-pair at the receiving node. Further on, let $tx_r(p)$ and $ack_r(p)$ be the times needed to transmit the p-th packet on the r-th try and its corresponding acknowledgment respectively. Then the round trip time is calculated as follows:

$$
\begin{aligned}
RTT(p) &= T^{\mathrm{req}}(p) + T^{\mathrm{resp}}(p) + \Phi_p \qquad \text{with} \\
T^{\mathrm{req}}(p) &= \sum_{r=1}^{R_p^{req}} (\mathrm{DIFS} + \delta_r^{\mathrm{req}} + tx_r(p)) + \mathrm{SIFS} + ack_r(p) \\
T^{\mathrm{resp}}(p) &= \sum_{r=1}^{R_p^{resp}} (\mathrm{DIFS} + \delta_r^{\mathrm{resp}} + tx_r(p))
\end{aligned}
$$

In real systems, end-to-end latency is a combination of different individual delays that occur at various layers. It can be divided into the following areas:

- Propagation & transmission delay
- Interference & retransmission delay
- Contention delay and
- OS overhead & application processing delay

In the following four paragraphs we take a look at each item in detail.

First of all, the propagation and transmission delay defines the lower bound for any given frame and distance. But since we are usually dealing with comparatively short distances, the propagation delay has a negligible impact on delay in single-hop wireless LAN transmissions (approx. 0.3 μs per 100 m) . Obviously, the transmission delay differs for different packet sizes and transmission rates. It also changes for different standards of the IEEE 802.11 family because e.g. the length of the physical header (preamble and PLCP) is smaller in .11g than it is in .11b. At the sender, the transmission rate for each individual packet is determined by a rate adaptation mechanism. This mechanism can e.g. take values like the packet size as well as the history for

previously transmitted packets into account. Auto Rate Fallback (ARF)[72], its adaptive version called AARF, or Adaptive Multi-Rate Retry (AMRR)[73] are well-known algorithms for rate adaptation. These algorithms perform differently under various network conditions, thus influencing packet delays. Rate adaptation algorithms usually do not distinguish different application requirements but work on a per-node level.

Furthermore, unicast data transmission in WLANs is provided by a reliable service, where each frame must be acknowledged by the receiver. If the sender does not receive an acknowledgement for its transmission because either the data frame or the acknowledgement were lost, the packet is automatically retransmitted. Packets can be lost because of interference or because of collisions with frames from other stations. Interference can come from other transmissions as well as other factors like attenuation, fading, shadowing, reflection, scattering, or diffraction. Most of the newer WLAN chipsets support multi-rate retries which allows a fine-grained control over these retransmissions. This means that the transmission rate of each subsequent retransmission can be controlled by the rate adaptation algorithm instead of sending a retransmission at the same rate over and over again. For retransmissions of a single packet the transmission rate is usually decreased with the number of retries to increase the chance of successful reception at the receiver.

Data packets may also experience additional delay from channel contention or congestion. The WLAN protocol specification provides for a contention period to arbitrate channel access when it is likely that multiple stations are ready to transmit. In this contention period each individual station waits a random amount of time before it is allowed to transmit and the station with the shortest waiting time wins access to the channel. While WLAN's contention phase improves channel utilisation for the network, individual frames experience an additional contention delay. Furthermore, if a station likes to transmit a data packet and determines that the channel is currently in use by another station, it has to wait until the channel is free.

Finally, data which is sent and received by an application also has to spend some time inside the operating system kernel. It has to make its way through the protocol stack until it is made available to the corresponding program's address space. There, it waits until the program is scheduled by the operating system to run on a CPU.

Composition of latency on real hardware

We performed initial measurements to determine the magnitude of individual components that contribute to latency on real hardware. We used two 1.8 GHz Pentium M

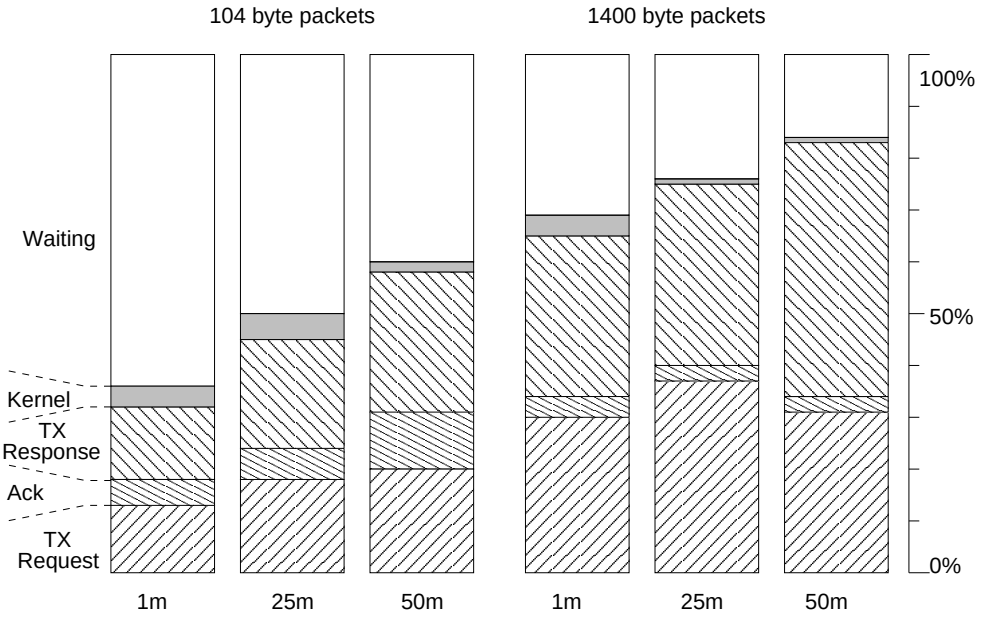


Figure 3.3.: Average time spent for different tasks (in percentage of RTT)

notebooks equipped each with an Atheros-based WLAN card[1] which we used in the 2,4 MHz ISM band and 802.11b/g. We measured data traffic between both notebooks with ICMP traffic of various size at distances of 1 m, 5 m and continuing in 5 m intervals up to 50 m. Our measurement took place outdoors in an urban environment with direct line of sight between both antennas. We modified the Atheros WLAN driver in the Linux kernel to give us queuing, transmission and reception events of WLAN frames with additional information about frame type and content. A high-resolution time stamp is then added to each event using the CPU's internal time stamp counter (TSC). More details on our measurements and its setup can be found in Section 5.1.1.

Unfortunately, current off-the-shelf WLAN hardware partly handles the WLAN protocol directly on chip without exposing all relevant performance details to the operating system. Hence, instead of using our delay classes previously defined on page 49, we have decided to directly show the classification which we use in our WLAN driver. Table 3.3 shows three transmission times for request, response, and acknowledgement as well as the operating system kernel overhead and the waiting time. It shows the relative shares of round trip time for each item which were calculated from taking the average values of 1000 request/response pairs in an uncongested network. The waiting time combines all congestion and contention delays as well as protocol overhead in the form of inter-frame spaces. From our measurement, we can see that the general

[1] 3Com Wireless 802.11 a/b/g PC Card (type 3CRPAG175B)

overhead for small packets is significantly larger than for larger packets. For distances of up to 25 m this overhead contributes 50 % or more to the total round-trip time between two stations. We also notice that the proportion of overhead gets smaller with increasing distance between our two notebooks. This is because the WLAN driver uses smaller transmission rates to cover the larger distances to achieve a sufficient signal-to-noise ratio at the receiving end. We measured a kernel overhead of about 30 μs which is fairly constant throughout different test measurements at various transmission speeds, node distances, and packet sizes. From our results we see that small packets could greatly profit from protocol optimisations and QoS priotisation efforts that aim at reducing contention overhead.

Related Work on Latency

During the last years and in the course of the development towards mobile networking, wireless technologies and especially the WLAN standards IEEE 802.11a/b/g have been extensively discussed in related work. However, nearly all papers that looked into the performance of WLAN focused on measurement of throughput and packet loss only. Examples are [74, 75] where latency was considered only as part of the bandwidth-delay product. Other related work focuses on packet loss in WLANs. Hlavacs et al. [76] state that packet loss shows a bursty behaviour and a dependence on losses of the near past for MPEG4 video streaming over WLAN. They use an extended Gilbert model to describe the packet loss. Carvalho et al. proposed a new packet loss modell for IEEE 802.11g in [77] and compared it with the Gilbert-Elliot model. Also Bianchi et al. [78] measured packet loss in mesh networks. Lenders, Wagner and May measured an outdoor wireless ad-hoc network in [79]. Li et al. [80] use the received signal strength indicator and the average wireless link capacity to predict the frame rate for streaming video over WLANs. Aguayo et al. [81] measured a 38-node urban multi-hop 802.11b mesh network. The focus of this work is on packet loss and the effect of multi-path propagation. Issariyakul et al. [82] developed an analytical model for batch transmissions in multi-hop wireless networks. Their work includes multi-rate transmissions and retransmissions. They also look at end-to-end latency in order to determine the performance of TCP in such environments and evaluate their model with NS-2.

Thus, while most related work focuses on measuring or increasing throughput, interactive applications often can cope with smaller bandwidths but have rather stringent delay requirements. The delivery of their data packets is time-critical because delay has an immediate effect on the user's perception of the performance of the application and of the network. Discussions on latency or jitter are mainly found in reference to

handoff-latency in infrastructure WLANs or when proposing new QoS schemes.

3.1.4. Capacity

Based on the traffic pattern for an interactive latency-sensitive application, we can give an estimate of how many stations can operate within a single collision domain before the wireless channel gets saturated. This theoretical analysis can give us an idea of what to expect in a simulation or a measurement and allows us to determine a capacity estimate that we can expect from a given wireless technology.

On page 49 we caculated the minimum round-trip time in a wireless LAN for a request/response pair. We now take this idea one step further and create the WLAN time budget model to determine the maximum number of stations that can operate successfully in a single broadcast domain. Given the traffic characteristics of a latency-sensitive application and a fixed transmission speed, we can calculate the total amount of time that a station occupies the channel either in sending data of its own or receiving it from another station. By taking additional WLAN protocol overhead such as inter-frame spaces, WLAN acknowledgements, collisions/retransmissions and the contention period into account, we can calculate the total usage time of the medium for a single station. We then add additional stations using the same traffic characteristics until the wireless medium is fully saturated to determine the maximum number of stations.

While calculating the transmission times and the protocol overhead is straightforward, other parts of the WLAN time budget model are not. Hence, we take a closer look at how we determine the duration of the contention period and the number of collisions/retransmissions in our model. Before each transmission, every station determines a random number of waiting slots within the contention window. Because the WLAN protocol assumes that all random numbers are equally distributed in the network, we are able to use half the number of slots in the contention window as an average for all transmission. Still, this approach has two flaws. First of all, the contention window may be adapted by each station after a collision or a successful transmission on the network. This behaviour of the WLAN protocol cannot be reconstructed by a simple stateless model that we use in our approach. But because we aim at finding a capacity estimate rather than the theoretical upper bound, we believe that our simplified approach is still valid. Secondly, no contention phase is required if a station encounters an unoccupied channel. But as we aim at saturating the wireless channel, the second problem can be neglected.

Considering collisions and retransmission in our model is essential because WLAN,

like any carrier sense multiple access network, suffers from substantial performance degradation under high traffic loads.To calculate the number of retransmissions, we use a close-form approximation for collision probability in WLANs that was developed by Tay and Chau. In [83] they show that the collision probability p is

$$p\frac{1-p-p(2p)^m}{1-2p} = \frac{2}{W}(1+\frac{2}{3}n)\frac{n-1}{n}$$

where n is the total number of stations, W is the slot size and 2^mW is the maximum size of the contention window. This formular also shows that the collision probability p does not depend on packet size or transmission speeds. For our analysis we use a fixed setting with m=10 which corresponds to a maximum contention window size of 1024 slots which is a common values for WLAN networks. The WLAN time budget model calculates the collision probability for each frame and retransmits it if necessary.

Figure 3.4 shows the results of our model for the multi-player game 'Counter Strike' with the traffic characteristics presented in Table 3.2 as well as the collision probability. It shows that IEEE 802.11b can support a maximum of less than 16 players while it's faster successor IEEE 802.11g can take about 36 stations under ideal conditions. We can also see that newer WLAN protocols can accommodate a greater number of players even at lower transmission speeds (e.g. 11 MBit/s vs. 6 Mbit/s). This is because the newer WLAN standard has a smaller overhead, mainly because the comparatively large MAC header is transmitted using a higher transmission speed. This modification has an significant impact on applications like Counter Strike that send many packets. Figure 3.4 also shows that the collision probability does not exceed 0.5 which is the limit for the approximation as given by Tay and Chau[83].

Our theoretical model does not account for a number of other factors such as propagation delay, routing overhead, interference and additional traffic from other nodes as well as overhead from optional protocol features such as RTS/CTS. Furthermore and due to collisions and congestion delay, we are not able to determine if all frames are received in a timely manner that is suitable for our chosen application. Together with our previous remarks, the results of this theoretical analysis should therefore be considered as an optimistic estimate of an upper bound of the maximum number of players. Based on our initial results, we propose two strategies to increase the number of users. First of all, the wireless network capacity can be improved if the size of the collision domain is reduced. This can be achieved by using topology control mechanisms such as clustering or by reducing the transmission ranges of nodes so that a frame only reaches the receiving node. With newer technologies like the new 802.11n wireless LAN standard, advanced methods like beam forming are also available. However, these methods focuses on short range single hop communication only. If a larger group of users is distributed over a greater area, data needs to be transmitted

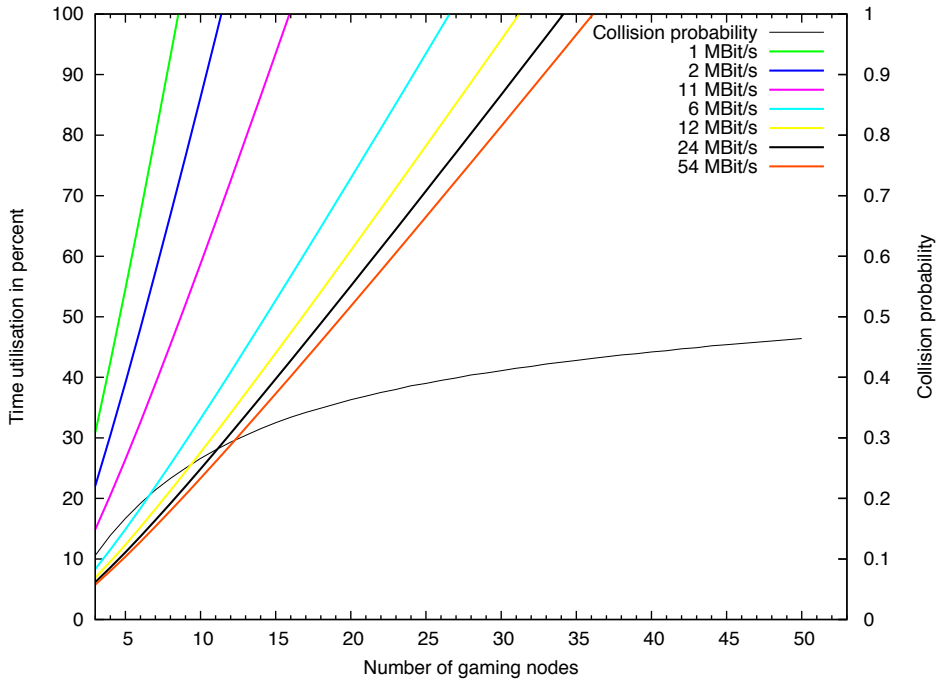


Figure 3.4.: Theoretical analysis of channel occupation time for Counter Strike

over longer distances. So secondly, methods should be devised that reduce the amount of data to be transmitted, e.g. by aggregating traffic between distant nodes or clusters.

3.2. Mobile Ad-hoc Networks

Wireless data networks haven been in use for more than ten years. Well-known and established radio technologies in this field are Wireless LAN (WLAN) and Bluetooth as well as General Packet Radio Service (GPRS), Enhanced Data Rates for GSM Evolution (EDGE), Universal Mobile Telecommunications System (UMTS), and High Speed Downlink Packet Access (HSDPA). They offer wireless communication for mobile devices such as laptops, mobile phones, personal digital assistants (PDAs). All wireless data communication follows one of two major paradigms.

In a centralized or infrastructure mobile network, a central entity often called base station or access point manages the entire network. It often also provides a gateway to existing wired networks such as the Internet or the public switched telephone network (PSTN) . If a centralized channel access scheme is used, this central entity also provides channel arbitration for the wireless network. In an infrastructure network, all

data is sent by or through the central entity, meaning that the base station is either the receiver or the sender of every packet on the wireless channel. Figure 3.5(a) shows an example of infrastructure networks with a base station at the center and some mobile nodes.

Wireless networks that work without any central entity are called decentralized or ad-hoc networks. Here, all nodes communicate directly with other nodes. And because there is no infrastructure which maintains the network and at the same time defines its physical location, ad-hoc networks may be transient and are created as needed either automatically or through manual intervention. Because wireless nodes in ad-hoc networks do not need to maintain a link to a central access point they have a greater degree of freedom in terms of mobility. Thus, the term mobile ad-hoc network or MANET is often used. Like infrastructure networks, MANETs may also include gateways to other, e.g. wired, networks although such gateways have no special role here. Figure 3.5(b) shows an example of a mobile ad-hoc network where all nodes can communicate directly with their neighbours. We decided not to show all possible links in this example to differentiate between MANETs and their multi-hop variant.

In the literature, the term 'mobile ad-hoc networks' often implies mobile multi-hop ad-hoc networks. As a single node has only a limited radio range the desired communication partner may currently not be in range. Nodes in multi-hop ad-hoc networks relay data to distant receivers by making use of other nodes in the network to forward data. This behaviour can greatly extend the communication range of a mobile node and allows for larger mobile ad-hoc networks. Multi-hop networks require cooperative behaviour of participating nodes to forward data in order to work. It is generally assumed that because any other node will forward my data that in return I will be willing to forward data from other nodes as well. Figure 3.5(c) illustrates a mobile multi-hop ad-hoc network.

In this thesis, we focus on mobile multi-hop ad-hoc networks (MANETs) which have some unique characteristics. From the network point of view, communication in MANETs is more difficult compared to wired networks. As with all radio networks, interference may cause data to be corrupted or lost completely during transmission, requiring a retransmission. With MANETs, mobility adds another layer of complexity to the system which impairs communication, reachability and connectivity further. But mobile ad-hoc networks also offer new possibilities. They promise efficient communication directly between users as a flexible, more natural form of communication instead of taking a detour through an access point. This type of communication, at least in principle, offers lower latency and higher throughput for mobile users. This also means that user mobility is not restricted anymore to areas where wireless infrastructure has been set up in order to communicate with each other. MANETs can be

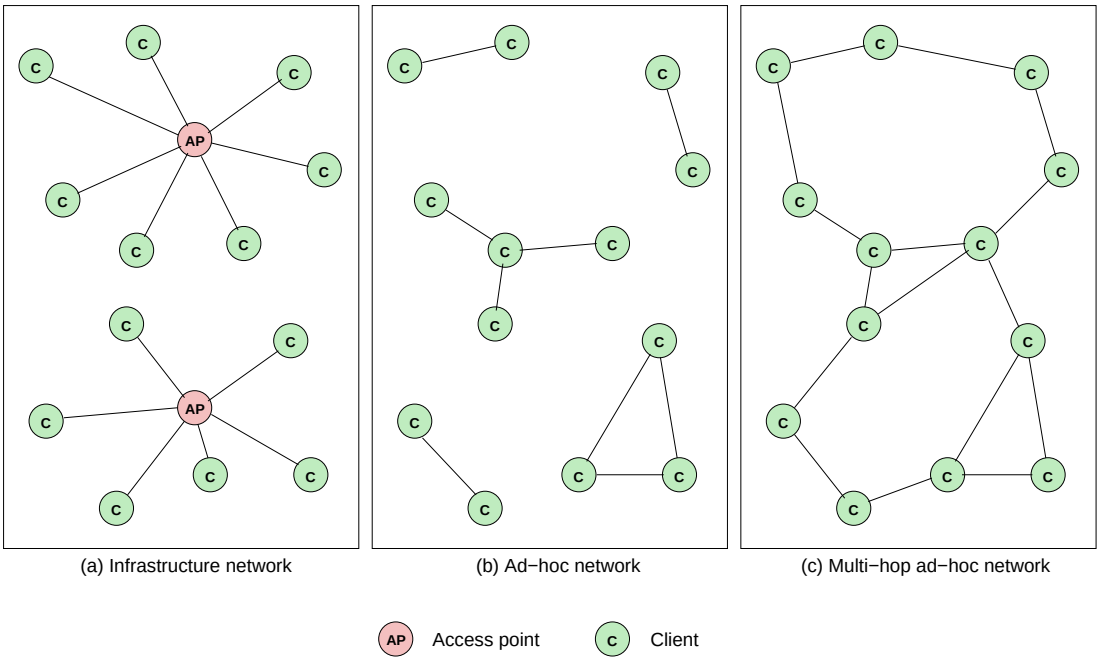


Figure 3.5.: Different types of mobile wireless networks

created in a sporadic manner at any time and anywhere. They require no pre-exisiting, pre-planned infrastructure. MANETs that operate over toll-free frequencies, e.g. like WLAN in the ISM band, can be used without any charges. Furthermore, they can also be used in combination with infrastructure networks to extend the range of an access point or to connect two distant MANETs.

The different network design and peer-to-peer multi-hop communication in MANETs lead to various new challenges. First of all, every node is considered both host and router at the same time because they forward traffic from others. In that lies a fundamental difference from the Internet philosophy which explicitly distinguishes between router and hosts. If every node acts as router in the network and is able to change routing decisions for other nodes, new security and privacy implications have to be considered. Furthermore, we already discussed that mobile MANETs require a cooperative behaviour of nodes to be able to forward data efficiently through the network. Nodes that use the MANET but on the other hand refuse to forward data from others reduce connectivity and therefore spoil network performance. Handling of these selfish nodes that provide limited or no service for others is another challenging task in mobile ad-hoc networks. In related work, several approaches to deal with unsocial behaviour can be found. With monitoring-based solutions, nodes are watched by their neighbours to detect unsocial behaviour. Other solutions establish a reputation for nodes or exchange virtual money for services. More on selfish nodes can be found

in [84]. Moreover, MANETs need to be self-organizing because they operate without any infrastructure. This means that traditional address allocations methods like the dynamic host configuration protocol (DHCP) cannot be used because they operate with a centralized architecture which rules out any address conflicts. Instead new and fully distributed approaches have to be found to determine unique addressing information for individual nodes. A common problem with node mobility in wireless networks is that it can lead to intermittent loss of network connectivity. In MANETs, the network itself can be mobile if all nodes are on the move. This additional freedom may result in a split network where two or more groups of nodes lose all connections with other groups but are still able to communicate among each other. Moreover, some of those groups may rejoin later on and reestablish connectivity with each other. An application in MANETs that wants to deal with such occurrences faces difficult synchronisation problems.

In the area of MANETs, several other related research fields emerged during the last couple of years. In the following, we take a look at three main fields, discuss their specific characteristics and what distinguishes them from the MANETs used in this thesis. First and foremost, wireless sensor networks (WSNs) comprise smaller devices with sensing and/or actuating capabilities, limited computational power and scarce energy reserve. They often send sensor data towards one ore more data sinks. WSNs are, for example, used to monitor forest fires, for industrial plant control, in health care and for military applications. The difference between MANETs and WSNs is that wireless sensor networks often use smaller devices without any screens or input devices and that they operate in a single administrative domain. For interactive applications, wireless sensor networks are not well-suited because they often offer limited throughput. On the other hand, in industrial or military applications, WSNs are often required to provide latency-sensitive services.

Vehicular Ad-hoc Networks (VANETs) are a specialized form of mobile ad-hoc networks that deal with car to car communication. VANET nodes can tap into the electrical infrastructure of the car and therefore do not have the same energy constraints as other mobile nodes. While MANETs often assume maximum speeds of 15-20 m/s, VANET nodes travel up to 60 m/s or higher. If two cars on the same road communicate with each other but drive in opposite directions, the differential speed is doubled and the window of opportunity for communication is small. Because VANET nodes may have information about their current location (e.g. through GPS) or means to determine direction and speed of travel, routing protocols for VANETs are often position-based. Example scenarios for VANETs are traffic jam alerts, brake assistants, and autonomous driving.

Delay tolerant networks (DTNs) are also similar to mobile ad-hoc networks. They fo-

cus on scenarios where nodes are spread out over a larger area with only intermittent communication opportunities between nodes and usually no continuous connectivity. Like in MANETs, data is passed along in a multi-hop fashion, however, it may take much longer time to reach its destination. In contrast to MANETs where it can take up to several seconds for data to arrive at the receiver, DTNs deal with delays of minutes, days or even longer periods of time. Thus they break the direct end-to-end philosophy of the Internet and therefore are not able to rely solely on the Internet Protocol. Hence, DTNs specify their own protocol stack that transmits data in so called bundles. Also, because of the different nature of DTN networks, methods and protocols from MANETs cannot be applied directly to DTNs. Example of delay tolerant networks are an inter-planetary network and distributed environmental monitoring. Due to their nature, DTNs are not suited for latency-sensitive or interactive applications.

3.2.1. Routing

In a wireless infrastructure network where a base station communicates with a mobile node or vice versa no routing is required. While they regularly employ methods to deal with node mobility between base stations, data transmission is usually directly point-to-point. In MANETs where multi-hop communication is possible and desirable, the network needs to find a suitable route from sender to receiver. However, and unlike the Internet, node mobility, radio interference and other obstacles make MANET links more volatile than wired links. In wired networks, links are usually stable for a long time and routing changes occur mainly because of manual intervention such as a reconfiguration of the network or in case of failures. In MANETs, mobility plays a more prominent role and any routing protocol has to consider that network links are created and lost on a regular base due to mobility at the sender, the receiver or any intermediate node along the way. This characteristic is the reason why existing Internet routing protocols such as BGP or OSPF should not be applied directly to MANETs. Specially designed new routing protocols for MANETs can discover and maintain routes faster and with less overhead. They have to fullfil two major tasks. They need to find a suitable route through the network to a given destination (route setup) and they have to maintain that route e.g. in case of link failures (route maintenance, route repair). Broken routes can either be rediscovered by starting a new route discovery process at the sender or the protocol can try to repair the route locally. With local repair, existing working parts of the route are retained while intermediate nodes try to find a suitable replacement for the broken link. For latency-sensitive application, routing plays an important role because it affects network characteristics such as latency, latency variation and packet loss. MANET routing protocols can be evaluated in three different

areas: Routing induced delays of data packets, storage and communication overhead and route quality.

Many routing protocols for mobile ad-hoc networks have been proposed during the last ten years. A common and established approach is to distinguish traditional topology-based and position-based routing protocols[85]. Position-based or geographical routing protocols exploit the idea that information about the position of mobile nodes can be used to route messages towards the destination. Firstly, a localisation service determines the position of the receiver. Then, usually a next hop is chosen which is closer to the destination. This process is repeated until the packet reaches its destination. By using the physical distance as routing metric, position-based routing protocols have to deal with local minima situations where a node is close to the destination but has no other neighbours that is closer. Position-based routing protocols are often found in vehicular ad-hoc networks where nodes are likely to be equipped with GPS receivers. They allow "geocasting" which sends data to all nodes that are currently in a defined physical region. Examples of position-based routing protocols are DREAM[86] and LAR[87].

Topology-based routing protocols can be divided into three subgroups. Proactive protocols use traditional routing strategies such as link-state or distance vector routing which are both used in the Internet. Link-state routing advertises routing information network-wide and as such is used mainly as an Interior Routing Protocol (IGP). Open Shortest Path First (OSPF[88]) is an example of a link-state protocol in the Internet. Distance-vector routing stores only vectors consisting of a next-hop address and an hop count for every destination. It is therefore suited for larger networks as well. An example of a distance-vector protocol is the Border Gateway Protocol (BGP[89]) which is the major exterior routing protocol of the Internet today. Proactive protocols gather and store information about the entire network regularly to have up-to-date information at hand whenever it is needed. Hence, the first packet to a new destination can be send immediately. However, as they store information about the entire network in the form of nodes and links (link-state) or vectors (distance-vector), there routing table size increases if networks get larger. Another point is that routing information is stored and maintained even if it is not used by the node. In MANETs, where mobility and interference may cause constant fluctuations of the network topology, both approaches have a significant communication overhead to keep their routing tables current and consistent with the actual network situation. An example of a proactive link-state routing protocols for MANETs is Destination-Sequenced Distance Vector routing (DSDV[90]). And Optimized Link State Routing (OLSR[91, 92]) is a proactive distance-vector based variant.

Reactive routing protocols discover routes on-demand whenever an application sends

data to a new destination. In contrast to proactive protocols, they usually have no general view of the network and maintain routes only as long as there are needed. This however leads to initial route setup costs in form of a start-up delay for new destinations that often is quite large compared to the round-trip time. To improve efficiency, most reactive routing protocols store additional routing entries to unknown destinations in a greedy fashion. This means that a node will look into routing packets that it forwards to find suitable new routes to other nodes. If a request for a new route is received by an intermediate node which already has a route to this destination, the request is answered directly and not forwarded to other nodes. Thus, routing information is shared among nodes in the network. Moreover, routing entries that are not used for a certain amount of time usually expire, so that stale routing information is not kept indefinitely. In MANETs, reactive routing protocols often have less communication overhead and fewer demand to store routing entries. On the downside, they have an initial setup cost for new routes. Examples of reactive routing protocols are AODV[93, 94] or DSR[95, 96].

Finally, hybrid routing protocols combine proactive and reactive ideas. For example, the Zone Routing Protocol (ZRP[97]) proactively exchanges routing information with nearby nodes that are within a given maximum distance, the node's local zone. Routes to nodes outside this zone are discovered in a reactive way. To operate efficiently, an appropriate trade-off between reactive and proactive elements need to be found. A common trade-off, however, is difficult to find for mobile ad-hoc networks that have different mobility characteristics and traffic pattern. Obviously, hybrid routing protocols are more complex than reactive or proactive approaches.

All previously discussed routing approaches treat all nodes equally. Hierarchical routing protocols for mobile ad-hoc networks differ in that they introduce multiple layers, so that, e.g., well-connected or more powerful nodes have more responsibilities than others. One common approach is to create routing backbones which are used to forward long-distance traffic in the network. A routing decision can then be made locally and only takes the traffic to the nearest backbone node which then makes a global routing decision. CEDAR[98] is an early routing algorithm for mobile ad-hoc networks that works according to this principle. It relies on an approximated minimum dominating set approach to calculate a core of nodes which are used to forward traffic. CEDAR also offers some support for Quality of Service. Wu[99] proposed another routing approach that uses a dominating set algorithm.

An overview of position-based routing protocols can be found in [85], topology-based protocols are discussed in [100]. Besides the aforementioned approaches, several other areas exist that deal with routing in mobile networks. In wireless sensor networks, routing protocols are often designed for low energy consumption or for special net-

work topologies that are used for WSNs, e.g. a sink-based topology. Another set of routing protocols deals with multicasting in mobile ad-hoc networks. Although some ideas such as energy-efficiency or multicast are likewise valid for latency-sensitive applications, we consider discussion of these issues out of scope of this thesis. Later on, we will discuss routing protocol features as part of our work on Quality of Service in Chapter 5.

3.3. Distributed Communication Architectures

The design of a communication architecture for a distributed systems is based on several different parameters. First of all, it needs to meet application requirements and offer suitable performance. It should also be scalable to be deployed on a larger scale then previously intended. In case of errors or failures, redundancy and resilience are additional factors that should be taken into consideration. Finally, its ease-of-use and simplicity aids in implementing and managing a distributed system.

In this section, we discuss different communication architectures. We start by looking at existing architectures in today's Internet. We then move on to published architectures for mobile ad-hoc networks that deal with mobility as well as the fact that any node can disappear from the network because its power source is drained or because of shadowing effects. Lastly, we discuss a specific implementation of these architectures for multi-player games. We continue this topic into the following section where we will present our zone server architecture for mobile ad-hoc networks.

3.3.1. Existing Architectures

Communication in a distributed environment is often designed according to either of two major paradigms: Client-server or Peer-to-Peer. A client-server model consists of a service provider (server) and one or more service requesters (clients). The server is usually well-known to the client either because it was directly defined by the user or looked-up through a well-known directory service. Schollmeier defines a Client / Server network as "*a distributed network which consists of one higher performance system, the Server, and several mostly lower performance systems, the Clients. The Server is the central registering unit as well as the only provider of content and service. A Client only requests content or the execution of services, without sharing any of its own resources.*"[101]. Thus, in a client-server architecture the role of provider and requester is clearly defined.

Not so with Peer-to-Peer (P2P) models where every participating computer can be service provider and service requester at the same time. Steinmetz and Wehrle [102] refine a definition from Oram[103] and says that a Peer-to-Peer system is defined as "*a self-organizing system of equal, autonomous entities (peers)* [which] *aims for the shared usage of distributed resources in a networked environment avoiding central services*". In other literature, terms like decentralised, scalable, and fault tolerance are often used in relation to P2P systems. Singh[104] described Peer-to-Peer as a way to support "*richer models of interaction than client-server*", being a "*federation of equals*". He also characterized Peer-to-Peer as a symmetric client-server model because each peer can query others. Figure 3.6 shows a classification of distributed systems architectures.

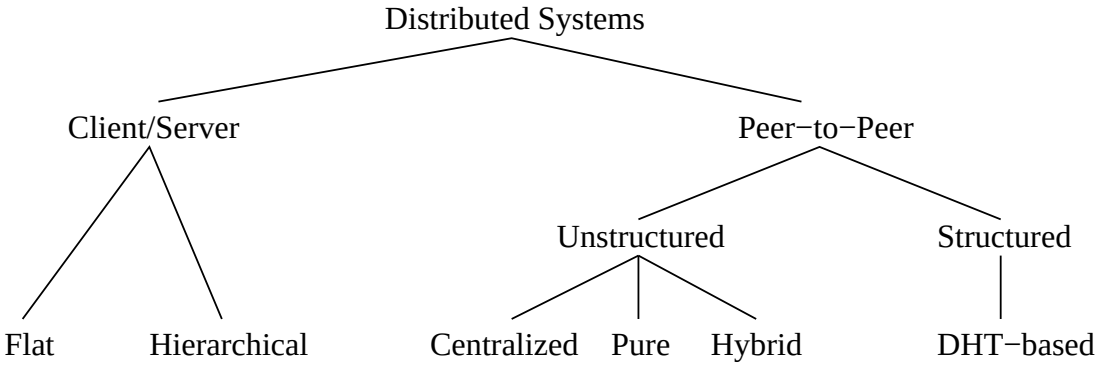


Figure 3.6.: Classification of Distributed Architectures

The flat client-server architecture is the most simple approach containing a single server which serves a group of clients. Most of the well-known internet services such as the web (HTTP[50]), file transfer (FTP[105]), email (POP3[106], IMAP[107]) or remote shell (SSH[108]) operate according to this principle. Another variant is the hierarchical client-server architecture where data is distributed among many servers in a structured hierarchical order. By using multiple servers, the hierarchical client-server approach can store much more data than a single server entity and provide service to a greater number of clients because the load can be shared among the servers. But with multiple servers finding the server that stores the data that a client is interested in gets more difficult. One popular access strategy for a hierarchical client-server architecture is a top-to-bottom search starting at the root node of the hierarchy. An example of such a system is the Domain Name System (DNS[48, 49]). Another example for a hierarchical client-server system is the use of server replicas to increase throughput. Here, data is copied from a single master server to a number of server replicas that can each support a number of clients. The client-server architecture has several advantages. It

is easy to develop and to manage, maintain, and secure. On the other hand, it provides a single point of failure and to a certain extent lacks scalability.

As can be seen in Figure 3.6 a variety of Peer-to-Peer architectures exist. The most prominent differences between them are the ways in which content is addressed and how lookups are done in the network. Unstructured networks can be divided into three types. They all share the property that they have no general indexing scheme. Centralized P2P systems employ a traditional server as central indexing facility for the network. All content is registered at this entity and lookups are centralized. After the data has been localized, the actual data transfer is done directly between peers without any further involvement from the server. An example of a centralized Peer-to-Peer system is Napster. Naturally, this architecture suffers from the loss of its server in the same way as the client-server approach. In contrast to centralized systems, pure P2P architectures do not rely on any kind of central entity. Such a 'pure' network consists of equal peers in a flat hierarchy. In a pure P2P network, any peer can be removed at any time without loss of functionality. Every peer is usually connected to a number of others peers but not to all of them. A bootstrap server provides an initial number of peer addresses for prospective nodes. An example of a pure P2P network is an early version of Gnutella (v0.4). Gnutella uses a flooding scheme to look for content in the network which obviously has some drawbacks. Freenet[109] is another example of a pure Peer-to-Peer network. Centralized and pure Peer-to-Peer systems were among the first P2P architectures that were developed. As we have briefly shown, both approaches have scalability issues when it comes to find out where data is stored in the network. Hybrid Peer-to-Peer networks combine both ideas in that they use special distributed indexing entities also known as super-peers. Here, each peer is connected to the network through its super-peer which creates an additional and dynamic hierarchy layer. Super-peers usually have a number of leaf peers and interconnect with other super-peers directly like in a pure P2P network. This setup eliminates the problem of a centralized master and at the same time drastically reduces the number of nodes that have to be contacted for a lookup. Super-peers can be regular client nodes that are chosen because of their increased performance, good network connectivity, suitable position in the network or by any other metric. Like in a pure P2P network, any entity can be removed without affecting the network as a whole. If a super-peer is removed, the now orphaned leaf nodes either have to find a new super-peer or the network gives super-peer status to one of them. JXTA[110] or later versions of Gnutella (v0.6) work according to the hybrid Peer-to-Peer principle.

Finally, Peer-to-Peer networks that create their own overlay network according to a common scheme are called structured P2P networks. There are no central entities in structured Peer-to-Peer networks but every node picks their neighbours according to

a given scheme. Structured Peer-to-Peer networks often use distributed hash tables (DHTs) and a keyspace that is partitioned among all nodes. The result is an efficient and scalable lookup mechanism that also provides an excellent base for effective routing mechanism (key-based routing). A disadvantage of DHTs is that hash functions only support direct-match searches but not a lookup based on keywords that describe and/or classify the content. Much research has been done in the area of structured P2P networks. Some well known examples are Chord[111], Pastry[112], Tapestry[113], and Content-Addresssable Network (CAN)[114]. More on Peer-to-Peer networks can be found in [102] or [115].

3.3.2. Peer-to-Peer communication in MANETs

Mobile Ad-hoc networks are Peer-to-Peer systems by nature. Therefore, it is coherent to base communication architectures for MANETs on the same principle. In related work, there have been three different approaches to bring existing Peer-to-Peer ideas from wired into mobile ad-hoc networks. Firstly, the simple adaptation of P2P technologies for mobile networks on the application layer. Secondly, a close integration of P2P application layer protocols with MANET routing schemes. And finally, the replacement of traditional MANET routing protocols with P2P technology on the network layer. In the following we will briefly give some examples of related work in these areas.

Repantis and Kalogeraki [116] present an adaptive content-driven query routing mechanism and three different dissemination strategies (disseminate to immediate peers, to local peers and to local and remote peers). They use multi-level Bloom filters to store and lookup information about content in the network and send content summaries to apropriate peer nodes. Goel et al. [117] proposed the use of Tornado coding to expedite data dissemination in a mobile ad-hoc network. Data is split up into coded segments and can be downloaded from different peers at different locations over time. If a sufficient number of segments has been received, the original file can be reconstructed at the receiver. Kellerer and Schollmeier[118] discussed the Zone Based Peer-to-Peer (ZBP) architecture which provides application-layer proactive search in mobile peer-to-peer networks. Here, each node defines an area around itself as its zone and broadcasts content information within that zone. Searching in ZBP is done in the node's own zone at first. If no results can be found then nodes at the border of this zone are contacted which again search inside their own zone and sent the request along until a match is found.

Klemm et al. [119] proposed the Optimized Routing Independent Overlay Network (ORION), a P2P file sharing protocol for mobile ad-hoc networks. It combines ideas

from Gnutella (keyword search through flooding) and AODV (reverse path routing) to automatically provide valid routes if the search was successful. Schollmeier et al. [120] have a similar approach and combine Gnutella-style flooding and DSR in their Mobile Peer-to-Peer protocol. Pucha et al. [121] look at how distributed hash tables can be used efficiently in MANETs. They present Ekta, which combines Pastry and Dynamic Source Routing. An evaluation shows that an integrated P2P and routing algorithm at the network layer provides the best performance. Their idea is further refined with Ekta+[122].

Fuhrmann[123] takes this idea one step further and introduced a general P2P-based routing scheme for large scale, hybrid MANETs. The Scalable Source Routing algorithm is based on Chord and works like a classical routing approach and a structured peer-to-peer overlay at the same time. Fuhrmann showed that his approach outperforms existing DSR and AODV protocols in larger networks and that it scales to more than 100,000 nodes. His work extends ideas of the Iterative Successor Pointer Rewiring Protocol (ISPRP)[124] that was published by Cramer and Fuhrmann two years earlier. However, while both approaches are able to find routes in the network, they do not necessarily find short routes. This is because the routing decision is based only on key lookups and does not take information about the location of the node in the network into account. Additional information about P2P approaches in MANETs as well as a categorisation and discussion can be found in [125].

This brief list shows that Peer-to-Peer ideas from wired networks have been successfully brought to the mobile world. With some adjustments, they can offer the same or at least similar advantages in these environments in spite of limited or intermittent connectivity. Like in the Internet, the strong point of Peer-to-Peer networks is that they do not rely on a single centralized system like the client-server approach. This type of architecture is unsuitable for MANETs because nodes can be mobile and rely on radios, an interference-prone communication technology. Thus any server can simply disappear from the network point of view of an individual node at any time with major consequences for the application.

Nevertheless and despite the advantages of Peer-to-Peer architectures, some drawbacks remain. Compared to a centralized server, fully distributed systems are often harder to design and more complicated to develop when application requirements include global data storage or a common decision. An example of a global data could be an access control list that restricts access to some content. Here, synchronisation methods have to be devised to communicate updates to this list back and forth between network nodes. Also, strategies have to be developed how to deal with inconsistencies of global data between nodes. Solutions to these problems in Peer-to-Peer networks often carry additional penalties in terms of network overhead and additional delay. Peer-to-Peer

networks are efficient in finding content in huge networks as well as providing high throughput when delivering content. This however does not necessarily go hand in hand with low latency requirements of interactive applications.

3.4. Architectures for Internet-based Multi-player Games

We now take a look at existing multi-player game architectures that are in use in the Internet today or that have been discussed within the research community. In the Internet, many commercial and popular multi-player games follow the client-server model. Because of the known shortcomings of this approach, several proposals have been made by the research community to increase performance and scalability. Several ideas follow the idea of multiple servers to serve a larger numbers of players while retaining a centralized systems. Others discuss fully distributed architectures for multi-player games that can communicate efficiently and provide solutions that allow global game decisions in a shared environment. In the following we present a number of these approaches, discuss their advantages as well as limitations and show some examples.

3.4.1. Client-Server Architectures

Popular examples of client-server based mult-player games are Quake, Counter Strike, both directly controlled shooters or Command & Conquer, an indirectly controlled strategy game. The reason for using the client-server model is first and foremost the simple design of this architecture. With the client-server model the entire game logic can be implemented at a single entity which leaves the game client mainly as an intelligent input / output device that can also deal with network difficulties. Usually, a single game server continuously handles a number of tasks which are executed in rounds also called ticks or frames. The fact that game servers work in rounds must not be confused with round-based games because this architecture is used also for fast-paced directly controlled multi-player games. Game server rounds merely operate as synchronisation points for all players in the game and occur as often as twenty or more times per second. A common list of tasks of a game server during a single round are:

1. Receive updates (e.g. movements or actions) from players
2. Validate all updates according to the rules of the game
3. Resolve inconsistencies between players (e.g. concurrent actions)

4. Create a new global game state
5. Propagate this new game state back to clients

Besides these tasks, a game server may have additional responsibilities such as keeping game statistics, providing software updates for the game client, doing user authentication or billing. Figure 3.7 shows an example of how the first-person shooter Quake communicates. We use Quake as an example because it is documented, well-researched and is available with its source code. Furthermore as a first-person shooter, it has high demands on network latency. In our example, the server starts by sending a game state update to all clients to ensure that all game participants operate on a common set of data. This game state is the result of the previous tick that is not shown in this example. With Quake, each tick lasts 50 ms during which period the server collects updates from the clients. Because Quake is a first-person shooter, we used player movement and firing events as player actions. In our example, the game server now collects updates from all players and until the end of the current tick. All updates that arrive after that deadline are deferred to the next tick. The game server then computes an updated game state and propagates it to its clients. Other games operate in a similar fashion.

For multi-player games fairness is an important issue. It includes not only methods to prevent players from cheating but also ideas to adapt to individual network conditions that players experience. The general idea is to remove or reduce the influce of any negative network characteristic like packet loss or delay from the game so that only the player's ability affects the outcome of the game. With the client-server architecture cheating prevention and network compensation can be addressed because a single server has all relevant information available. By using a round-based scheme that orders actions according to their arrival time at the server the game basically provides a common time for all players. If player-generated time stamps would be used, a player with a low-latency network connection to the game server could find methods to cheat by adjusting the timestamps of his actions to his advantage. The round-based scheme makes this kind of cheating impossible because player actions are executed in the order in which they arrive at the server. Thus, a centralized game server solves concurrency issues by making a decision that puts all actions into a proper temporal order. And because the game server sends out updated game states only after player actions have been committed players are prevented from learning other player's moves in advance. However if player's actions are ordered the way they arrive at the server, players with higher network delay are at a disadvantages because their reactions to a new and updated game state from the server usually arrive later. To correct this unfairness, game servers sometimes use a latency compensation method which constantly monitors the round-trip delay for every player in the game. When a player action is received by

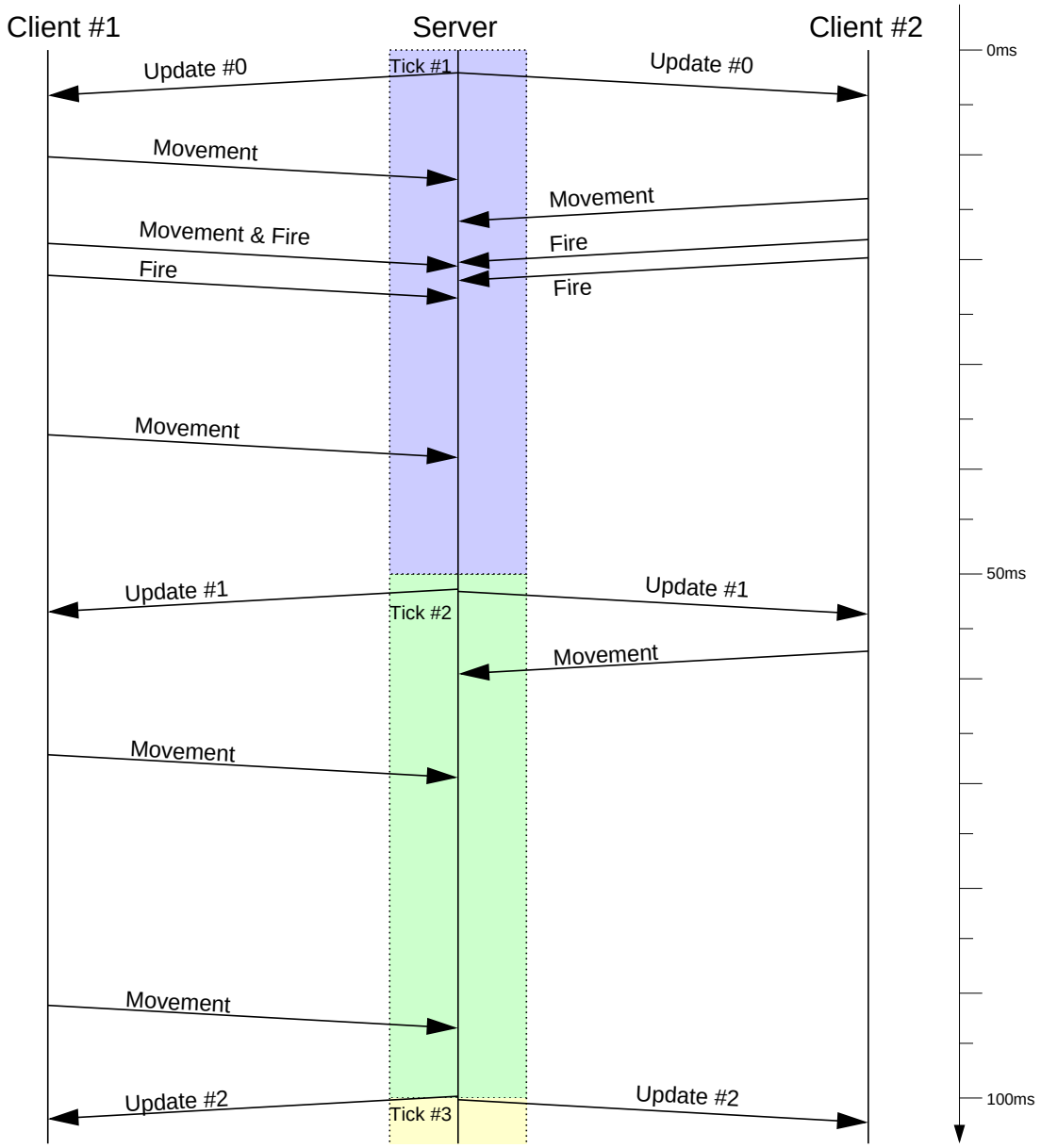


Figure 3.7.: A communication example of a multi-player game (Quake)

a server, the server subtracts half the round-trip time from the arrival time as an approximation of the one-way delay between player and server therefore negating any differences in network delay. Nonetheless while latency compensation provides for a fairer game, such features need to be designed carefully. A malicious player could exploit this scheme to his advantage by deliberately delaying some of his actions. Finally, game servers provide a central authority at which the rules of the game are enforced for all players. Upon reception of a player's action, the game servers needs to validate

this update to determine whether it constitutes an action that is allowed. E.g. if a player tries to move through a wall and such action is prohibited by the game, the server can detect it and take appropriate actions such as ignoring that particular move. Often, game clients already do their own checks based on their own copy of the game's rules to prevent updates being sent to the server who would throw it away anyway. Also, this local decision at the client is much faster than asking a server through the network thus providing a better service to the player. However, game servers often do their own checks anyway to prevent cheating by a modified client. Generally, the client-server architecture can address fairness and cheating quite well because it has access to the game state, updates and timing information from all players.

Depending on the complexity of the game and the total number of players, the actual size of the game state can be quite large. It is therefore expedient to reduce the updates before sending it to players to minimize bandwidth requirements and load at the game server. A number of different data reduction schemes are used today. First of all, the game state can be delta-encoded so that only elements that actually changed during a round need to be transmitted. Also, if an element did change, it can also be delta-encoded so that e.g. a position update of a player does not contain his absolute position in the game but his relative movement in reference to his last known position. Delta-Encoding is an easy and simple way to significantly reduce the bandwidth requirements of game state updates. Even if only a few bytes can be saved per player, the effect can be noticeable if the number of players is large and such updates are sent more than 10 times per second. But as delta-encoded game state updates build on each other, a lost update message can break this chain and possibly produce an invalid game state at the client. Hence, a method to resynchronize clients is used to deal with this problem, e.g. through retransmission of lost packets, by adding redundancy information to recover from lost packets or by regularly transmitting a full game state. Another approach to reduce bandwidth is based on the fact that a player does not need the complete game state but rather up-to-date information about players and objects in his vicinity that he can actually see or interact with. This region is called area of interest and is specific to each individual player. It often has a rectangular shape that is easier to compute than a circle or a sphere. Area of interest filtering is done by the game server and can reduce the load on the client and on the network significantly. However, it may have an impact on server performance when dealing with a larger number of players.

Traditionally, game clients play only a minor role in the client-server architecture with the game server handling most of the work. However, in the course of increasing requirements of multi-player games and improved performance of game clients they today perform various additional tasks. Examples are the previously mentioned checks to see whether a player's action is valid and in accordance with the game rules. Fur-

thermore, the graphical output of newer clients is decoupled from the game server rounds meaning that they are not bound to the server's frame rate anymore but can locally render 30 or more frames per second while position updates from other players continue to arrive at a rate of e.g. 20 frames per second. The position of players and objects between those updates are interpolated by the client. Interpolation algorithms for different games vary between simple linear methods and complex physical simulations of gravity and inertia. Also, clients can predict likely positions of other players if one or more updates are lost in the network. This dead reckoning scheme allows a continuous view of the game for the player even in times where the network is impaired for some time. If a new update finally arrives, the predicted and actual positions of a player or object are merged over time so that the user takes little notice of any brief network problem he might encounter. This behaviour is in contrast to traditional synchronisation schemes for distributed systems where often an exact representation of the data is required. Multi-player games, however, do not require full game-state synchronisation to all clients. Instead, games aim at partial synchronisation up to the point where it is possible to show the player a believable representation of the current state of the game that is fair and consistent from his point of view. More information on dead reckoning can be found in [126], [127], and [128].

3.4.2. Multiple Server Architectures

The client-server architecture is by design limited to a single centralised server which does not scale well with an increasing number of clients. For multi-player games this means that there is usually a maximum number of players that a single server can support depending on its hardware capabilities and the complexity of the game logic. In order to allow more concurrent players, several enhancements of the client-server architecture have been proposed. The Proxy Server Architecture[129] was proposed in 2002 by Mauve, Fischer, and Widmer. It introduces one or more proxy servers in addition to an authoritative game server that are placed in close proximity to the players. The authors mention the example that an Internet service provider could offer a proxy server for his customers. All proxy servers and the game server are interconnected by an overlay network through which they synchronize game states. And because they are not under the control of a player, they can take over some of the game server tasks to relieve the central server. Players make their decision to connect either to one of the proxies or the central server based on network delay, the number of connected players or any other metric. This approach in theory allows more concurrent players in the game as a traditional client-server architecture. It also can offer better performance for each player as a suitable close-by proxy server with low delay and load can be

chosen. However, multiple proxy servers add additional synchronisation overhead so that a good tradeoff between the distribution of tasks among the central server and the proxies on one hand and synchronization on the other hand needs to be found. By introducing a peer-to-peer like communication scheme among the servers, the proxy server approach retains its scalability while at the same time limiting the complexity and difficulties of a full-featured peer-to-peer approach.

Cronin et al. [130, 131] published the mirrored server architecture which is similiar to the proxy server architecture but does not rely on a single authoritative central server. Mirrored servers operate as equal partners and communicate with each other via a private low latency network using IP multicast. Thus, they need to be located at a common facitility. The authors propose the trailing state synchronisation scheme, an opportunistic algorithm with rollback functionality which is suited for low-latency interactive applications. Through the use of IP multicast, the mirrored server architecture creates a full-meshed peer-to-peer network that offers redundancy in case a server fails. Because all servers are located at a single facility and operated by the same provider, mirrored servers are easy to deploy and to manage. Hence, the mirrored server architecture is suited to simply replace an existing single server system. Compared to the proxy server architecture, mirrored servers remove the central server as a single point of failure and possible performance bottleneck. However, their scalability depends on efficient server to server communication. If every game event and player movement has to be propagated to all servers, processing of such update messages presents a new limit for the system. Even if area of interest filtering is applied to server-server communication, the movement of players in the virtual game world may undo much of the advantages of such a scheme if a player is assigned a fixed mirrored server.

Webb, Soh, and Lau[132] presented an extension to the mirrored server idea called enhanced mirrored server architecture (EMS) which allows peer-to-peer communication between game clients as well to reduce the load on the server infrastructure. The mirrors now act as trusted referees to validate the game state and to prevent cheating. In cases where peer-to-peer communication between clients is not possible, they can also fall back to their known role as mediators. EMS reintroduces a master server for authentication purposes. Because updates are directly between players, server synchronization is now less time criticial. Thus, the authors propose to use bucket synchronisation[133] instead of the processing intensive trailing state synchronisation scheme. Despite these improvements, the authors state that the “potential growth [of the enhanced mirrored server architecture] is still limited by its processing requirements as all mirrors must simulate the entire world.”[132]

Massively Multi-Player Games

All client-server architectures that we have presented so far have in common that they are not able to cope with thousands of concurrent users. To allow Massively Multi-Player Games (MMPGs), the communication costs between servers have to be reduced. Thus, MMPGs break with the idea that a player is firmly assigned to a fixed game server during the duration of the game. Instead a game server is responsible for a certain area of the virtual world and all players within that region. Players that move from one region to another have to switch to another server which is automatically handled by the game client. To allow a seamless transition, the involved servers start exchanging updates for a player as soon as he enters a pre-defined border region in the virtual world. This procedure ensures that the new server has up to date information about its new player early on which allows for a smooth transition. It also guarantees that players within the border region but on different servers can notice each other. First implementations of massively multi-player games pre-defined fixed server regions that could not be changed during the game. Later versions allowed dynamic assignements as well. By putting the focus on game world partitioning, this system has many advantages. First and foremost, it automatically includes area of interest filtering between servers because all players that are connected to a specific server are within the same virtual region. It also clearly defines border servers with which player updates and events have to be synchronized. E.g. for a regular rectangular-shaped region, a server needs to exchange data with a maximum of eight neighbours that server regions around it. From an architectural point of view, a single region in the game employs a traditional client-server model.

Massively multi-player games are scalable and can support thousands, sometimes tens of thousands of concurrent users. But despite their advantages, handling such a large number of players is still difficult because a single server can still support only a maximum number of concurrent users. And since players are usually not distributed evenly across the virtual game world, game server regions have to be properly planned in order to avoid congestion if a group of players gather at a single location. Here, strategies that are similar to network planning for mobile cellular network are used to create micro-regions to cover specific points of interest or hotspots like e.g. cities in the game. Newer and more advanced MMPGs divide the game world into small microcells and assign a number of microcells to a single server. This approach allows to dynamically move microcell responsibilities from one server to another in case a server is overloaded. Massively multi-player games that are based on this design have been successfully deployed in the Internet, some of which have been a huge commercial success. Although the architecture is basically a centralized client-sever design with some peer-to-peer elements between region servers, it provides an excellent scalability.

3.4.3. Peer-to-Peer Architectures

Multi-player games that employ a peer-to-peer architecture allow direct communication between players. As previously discussed, they are in principle scalable to thousands of nodes without having to invest in large server farms to manage them. However, the percentage of games using a peer-to-peer based design is very small. From a commercial point of view this architecture has been irrelevant so far because traditional architectures are easier to develop for and scalability issues could be addressed with multiple servers. Also for a company, it is easier to keep control of their game and make money through a server-based online service when their services are centralised. Neverthless peer-to-peer game architectures have been researched for the last ten years. They can be classified in three general areas: Fully-meshed, DHT-based and hierarchy-based approaches. In the following we give some examples of research in this field.

One of the first distributed architectures for games was MiMaze[134, 135], a fullly meshed real-time interactive peer-to-peer game that allows players to compete against each other. MiMaze does not require any server infrastructure but relies on cooperative clients that hold a shared state of the game. It is based on the real-time transport protocol (RTP) and uses multicast to communicate updates between players. Each player locally computes a game state based on his own actions and updates that are received from other players. A bucket synchronisation mechanism guarantees that all players are able to compute a coherent game state. The global clock that is needed for synchronisation is provided through the network time protocol (NTP). Jardine and Zappala[136] proposed a fully meshed game architecture for massively multi-player games that combines client-server and peer to peer communication which is similar to the mirrored server architecture. Events are directly communicated between players and only critical events are processed by the server to avoid cheating. This leads to reduced bandwidth requirements at the server which allows a larger number of concurrent players.

Lee and Sun[137] introduced a load-balancing peer-to-peer system for massively multiplayer games based on JXTA[110]. It comprises a central server for client authorisation and a number of so called subservers that manage a partitioned virtual world as well as area servers. The latter two run on the game client so that this architecture can operate with just one single server that is not on the critical path. If a partition gets overloaded by too many player, it can be dynamically subdivided into areas which are managed by area servers, creating a three level hierarchy. GauthierDickey et al. [138] presented a hierarchy-based event propagation scheme with interest filtering to reduce traffic between peers by using N-trees. This bottom-up approach is based on the idea

that events have a scope meaning a certain area which they effect. This scope defines how far up the tree an event is propagated and players with the smallest scope are the leafs of the N-tree.

Colyseus[139] is a game object manager that allows hundreds of players while keeping up with the tight latency requirements of Quake, a first-person shooter game. It employs peer-to-peer communication and uses Mercury[140], a DHT-based publish-subscribe system by the same author. With Colyseus, game objects can be pre-fetched according to an interest prediction algorithm. Therefore, short-lived objects that have high interaction potential with other players are distributed proactively. Iimura et al. [141] proposed a zoned federation model which splits the game world into zones and divides them across the peer-to-peer network. To provide redundancy, game data is stored across all nodes and can be found through a DHT. Modifications to the game state have to be made through the appropriate zone owner which is not efficient for fast-paced games. Hence, to be able to react quickly to any request, the zone owner caches data for his zone. The zoned federation model is designed as a middleware layer between the game application and the network.

In general, peer-to-peer networks have many advantages for multi-player games. They require little or no infrastructure and are therefore inexpensive to operate. The game state as well as the task of creating or modifying objects are spread out to all clients spreading the load across all players as well. Additionally as more players join the game, their own processing, storage, and communication capacities are added to the game. On the downside, timely synchronisation between player's nodes needs to be addressed. Generally speaking, game state synchronisation in a fully distributed system is a complex issue and not easy to achieve. An efficient game synchronisation scheme is thereby often dependant on the game and its specific set of rules requiring additional design and programming effort. However if a game synchronisation is specifically tailored, it can consider all necessary delay and consistency requirements of a multi-player game while at the same time demanding as little communication between the peers as possible. As game events such as player movements or modification of game objects are constantly generated, they need to be ordered across the distributed network even when they occur in different parts of the network. The ordering of events and the dissolution of concurrent events is a major part of any synchronisation scheme. For multi-player games, this behaviour is also essential to prevent cheating. Without any supervising entity like a game server, cheating is comparatively simple in peer-to-peer networks if no precautions are taken. A player could simply send fake game updates to other players.

3.5. The Zone Server Architecture

A multi-player game architecture for mobile ad-hoc networks needs to take the differences of such networks into account. As we have discussed above, network characteristics of MANETs are very different from those commonly found in the Internet. Hence, traditional and existing communication architectures from the Internet cannot be applied directly to mobile ad-hoc networks because they are not prepared to deal with new challenges like mobility.

We already discussed the problems that a centralized client-server infrastructure has in these networks as nodes may move around and network links get impaired or even break down completely. Depending on network density, mobility, and interference, clients could see intermittent but regular failures when communicating with their server. While any software for mobile ad-hoc networks has to deal with times where the network may be disrupted, a single server strategy cannot provide a suitable service if the server node is in the part of the network that is impaired. To counter local network-related problems, any communication architecture for mobile ad-hoc networks needs to eliminate single point of failures and provide a suitable amount of redundancy. Peer-to-peer networks are able to provide this redundancy. However, they scale so well in the Internet because nowadays clients have large performance and storage capacities as well as high bandwidth network connections. Thus, they are able to assist in managing the network as well as sustain high data rates to and from other peers. With mobile ad-hoc networks, we often have devices with limited capabilities regarding memory and CPU power. Furthermore, the famous 'last mile' or the last link to the client that is often the bottleneck in the Internet does not exist in MANETs. Here, all nodes use the same radio and have the same communication capacities whether they are located at the center or the edge of the network. Even worse and depending on the radio link quality and routing protocol, important links in the center of the network could also have fewer bandwidth than links at the edges. Thus, an architecture for multi-player games has to take network connectivity as well as the processing capability of mobile devices into account.

3.5.1. Overview

In 2003, we first published our idea of the Zone Server Architecture[142] which was further refined later on. It is tailored to the peculiarities for mobile ad-hoc networks and takes up ideas from hybrid peer-to-peer networks which combines direct communication between peers with client-server like but dynamic central entities. While our main goal was to create an architecture for multi-player games, it has no specific pre-

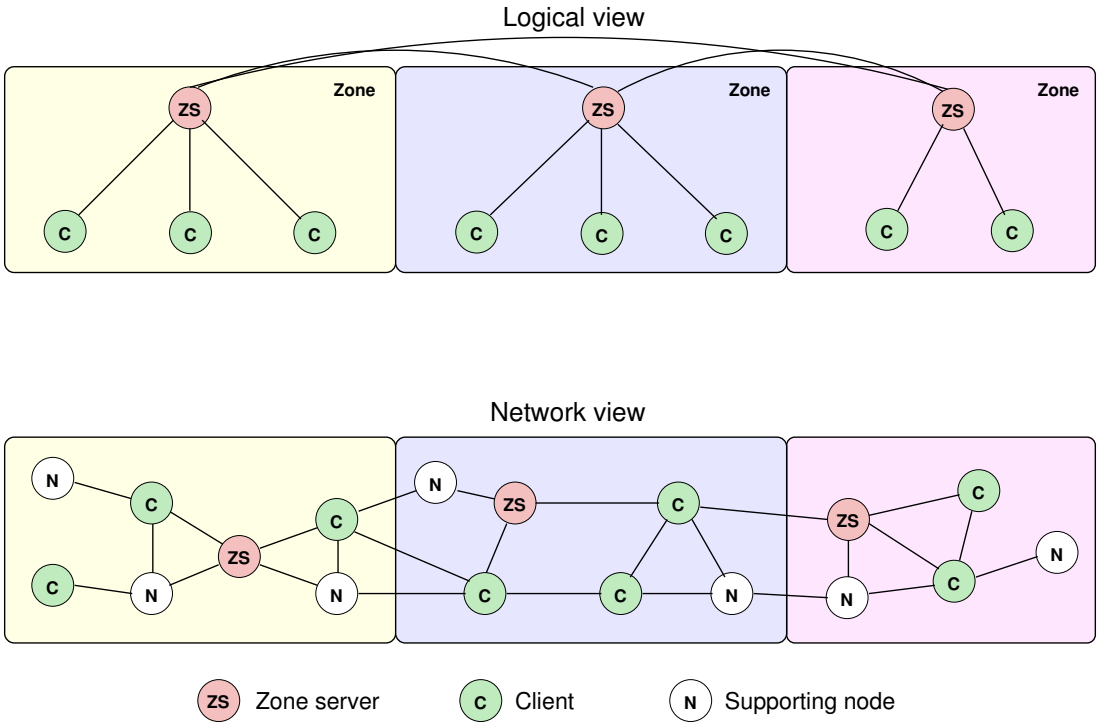


Figure 3.8.: The Zone Server Architecture

requisites regarding the type of application. It is therefore generally applicable however especially designed for low-latency applications with a larger number of users. Without loss of generality, we continue to use a fast-paced multi-player game as our example application. The Zone Server Architecture comprises multiple zone servers that each serve different areas of the network, their zones. A client connects to its local server which is responsible for the zone in which the client currently resides. Client and server both communicate in a traditional client-server sense. Zone servers on the other hand exchange information directly in a peer-to-peer fashion. Their role as zone servers could also be described as a cluster head for the local network region.

Figure 3.8 shows an overview of this multi-server approach for mobile ad-hoc networks. It gives an example with three zones and presents the architecture from two different points of view. The logical view shows three zones with their respective zone servers and the clients they serve. The zone servers coordinate with each other and exchange synchronisation data. Communication between zone servers is done by using direct multi-hop connections on the network level although the decision which data is passed on to which other server depends on the actual requirements of the application. As such the zone server synchronisation scheme should be defined individually for an application. The network view shows the same example but now with network links

between the nodes. It also contains additional supporting nodes that take part in the mobile ad-hoc network but do not run our application. The network view illustrates that zone servers can reside anywhere within their zone and that clients can be connected to their servers either directly or through a multi-hop connection. Zone servers do not form any kind of backbone but instead they utilize the mobile ad-hoc network as any other node would do. Synchronisation or any other data that is passed between zone servers can even be transmitted via a client node although its application would be oblivious to the data transmitted because this information is forwarded on the network layer and is not examined by the application itself.

Zone servers are selected by a server selection algorithm from the group of all clients. Supporting nodes are not considered for this task because we believe that the owner of a mobile device lacks the incentive to offer a service for others that only drains its own battery power but offers no advantage in return. This is in contrast to his responsibilities of forwarding data as a member of the mobile ad-hoc network where all nodes equally provide a service for the whole network. Furthermore, supporting nodes generally do not have the application software installed to provide a zone server service. The server selection algorithm determines suitable clients as zone server candidates. Such candidates should have the necessary energy and CPU performance reserves to assume zone server responsibilities in addition to their client role for their local users. In addition and in order to reduce unnecessary traffic in the MANET, zone servers should also be chosen for their good position in the network. Clients then connect to the closest zone server and a zone is defined as the area where all clients of a particular server are located. To be able to provide a good and timely service for latency-sensitive applications, clients and servers should not be far apart in the network. We therefore define an upper bound for the number of hops between a client and its server. The server selection algorithm then creates the necessary number of zones accordingly so that this limit can be met for most clients. Before we will describe the server selection algorithm in the next chapter, we now look at some general features offered by the zone server architecture.

3.5.2. Design

Because an ad-hoc network consists of mobile nodes that can move around, routes over fewer hops tend to be more stable than longer routes. Therefore, by bringing client and server close together, we can generally minimize network fluctuations which results in reduced latency and less chance for packet loss. For multi-player games, this means that player actions are transmitted by the client to its nearby zone server which can then make a local decision based on its current version of the game state. And because

clients are close-by, game state updates could be propagated by using broadcasts or restricted flooding to further reduce traffic inside a zone. However, this decision is not made by the zone server architecture but is left to the individual application. By putting clients and server close together, the zone server architecture keeps the critical path inside each zone and ensures scalability. This however moves some complexity to the server synchronisation if local decisions by two servers are conflicting with each other. But we believe that for example for multi-player games the resolution of such inconsistencies in the distributed game state is manageable because no strict data consistency is required. Also, synchronisation is a general problem for peer-to-peer architectures as discussed earlier.

Similarly to a client-server design, the zone server architecture adds some complexity to the server but keeps the client comparatively simple. In a network with several different kinds of devices, this approach has many benefits because the server role can be taken by more powerful devices. Furthermore by keeping the architecture similar to the well-known client-server design, existing applications can be adopted to mobile ad-hoc networks and existing knowledge can be used to design and program new software. Another essential feature of the zone server architecture is to provide redundancy in case of node or link failures. In a mobile ad-hoc network it is not possible to guarantee that a client can always reach a server at any given time or that it is even connected to the network for that matter. Still, multiple servers that are spread out across the network can provide the necessary backup functionality. If for example the middle zone server from the example in Figure 3.8 would fail or loose its network connection, its clients can connect to the left or the right zone server with a maximum of two hops.

3.5.3. Aggregation & Mobility

Partitioning the network into multiple zones does not only provide redundancy but has additional advantages. A zone server is able to aggregate application data before it is synchronized to other servers. Also, based on its knowledge about the game state, it can decide which information is important and should be forwarded immediately and which is less important or even redundant. Thus, the server is able to employ area of interest algorithms to further refine and reduce the data volume. Hence, a zone server is able to reduce synchronisation data between zones and save bandwidth. The magnitude of this reduction is of course dependent on the application. For mobile ad-hoc networks data reduction is important because synchronisation usually uses longer routes and therefore involves more nodes in the network. And a scheme that keeps most of the data transfer locally and aims at reducing long-distance bandwidth is generally considered to be network-friendly.

Any communication architecture for mobile ad-hoc networks also has to take mobility into account. The zone server approach supports mobility for servers as well as for clients. Both variants are in fact the same problem because any server movement could also be seen as moving all its clients at once. Generally, it is advisable for wireless clients to monitor all links to their neighbours as well as the connection to their server because they can detect upcoming impairments in advance. In doing so, they can proactively deal with network impairments before any noticeable effect on application performance occurs. In the Internet, many latency-sensitive applications already do connection monitoring to be able to compensate for any unexpected delay, delay variation or packet loss. Link monitoring in MANETs can give additional information about upcoming network impairments which is usually not required in the Internet as it is rather static and more stable. While being beneficial, the zone server architecture has no requirement for applications to do any link or connection monitoring.

If a client starts to move, often no actions are necessary at first because network links will only be affected a little bit and the application will continue to work. As the client moves further, existing network links may break down and new links are created. Here, the underlaying MANET network routing protocol kicks in to maintain a valid route to the server as data is obviously flowing back and forth. If the application is aware of these changes, e.g. due to monitoring as mentioned above, it can now start to look for an alternative server. As the client continues to move, the application may decide that it is best to switch to a new zone server. The client could do this by simply connecting to the new server and dropping its ties with the old server or it can use another, smoother transitioning method. As data between zone servers is synchronized, a client can be connected to two servers at once for a short period of time to ease server transition. This method ensures that the new server has all current information about the client at hand prior to its transition. In cases when the connection to the server is abruptly lost, a client uses the server selection algorithm to find a new server just like at the start of the application. However, with a sudden connection breakdown the client has to find a new zone server and connect to it which again has to make sure that it has up-to-date information available for this particular client. In MANETs and depending on the network conditions, the client may be not fully integrated for a couple of seconds. This means that in contrast to Internet applications, applications in MANETs have to be aware that these sporadic interruptions may happen at any time and deal with them accordingly. As more clients start to move around, the server selection algorithm may also decide that a client is capable and in a good spot to become a server itself. If this happens, the new server would join the peer-to-peer network of zone servers, update all necessary data for the application and then announce its availability to the network. Clients may decide to switch to this new server in the usual way. Finally, it should be mentioned that while the zone server architecture can deal with network partitioning

by letting the instances in both networks run individually, any eventual re-merge is difficult to handle because it involves the synchronisation of diverged and possibly conflicting application states. It is therefore the task of the application to handle these issues.

3.5.4. Cheating Considerations

An important problem of peer-to-peer networks is cheating. The zone server architecture has similar problems, because it uses a peer-to-peer design for communication between servers. Additionally, as servers are determined by selecting suitable clients, a malicious user can gain access to the server network. Therefore, suitable anti-cheating mechanisms are advised to prevent a malicious user from becoming a server. For peer-to-peer networks several such approaches have been proposed, e.g., by Kabus et al. [143, 144]. An overview of anti-cheating schemes was done be Webb et al. in [145]. While such methods against cheating can also be used for the zone-server architecture, there are some differences. One problem of peer-to-peer anti-cheating schemes is that they introduce an additional overhead to detect fraud. For example, the lockstep protocol by Baughman and Levine[146] requires all clients to send a hash value of their actions as a commitment message prior to the actual action message itself. While this approach helps with the look-ahead cheating problem in peer-to-peer networks, an additional round of commitment bears a time penalty that makes it unsuitable for fast-paced applications. With the zone server architecture, servers are responsible for detecting cheating just like a traditional server in a client-server system is. Moreover, exisiting anti-cheating schemes like the lockstep protocol could be used among zone servers. As zone server synchronisation can often be done asynchronously, it bears no additional time penalty for the clients.

Furthermore, the zone server architectures allows the participation of untrustworthy clients to participate as long as they are prevented from assuming server responsibilities. This is in contrast to peer-to-peer approaches, where all clients have to be taken into account.

3.5.5. Related work

The zone server architecture shows some similarities with previous mentioned approaches. The proxy server architecture[129] uses multiple proxy servers which are located near the clients to provide good performance and low latency. Similar to the zone server architecture, this scheme allows for direct communication between proxy

servers. However, it still requires a master server as a centralized coordination instance which is inapt for mobile ad-hoc networks. The mirrored proxy architecture[130, 131] also has some similarities as it uses multiple servers for better scalability. It abstains from using a central entity but requires IP multicast and a private low-latency network between the servers. While its idea is similar, it is especially designed to overcome the scalability issues of client-server systems in the Internet. Finally, area servers[137] that split up responsibilities for a virtual world are a common approach for massively multi-player games. If a player moves from one area to another in the virtual world, he also needs to connect to a different area server. The zone server architecture uses a corresponding idea which referes to the real network instead of the virtual world. Here, the network is split into different zones and real movement of clients may lead to the change of zone servers for the client.

3.6. Chapter Summary

In this chapter we discussed wireless LAN technologies and the impact of radio networks on interactive and latency-sensitive applications. We put special emphasis on fairness and latency issues and provided a theoretical analysis of the maximum number of players by using the multi-player game 'Counter Strike' as an example application. We have shown that latency-sensitive applications are at a disadvantage in WLAN networks when competing with traditional TCP traffic because of the per-node fairness property of WLAN. Furthermore and due to the lack of a mandatory quality of service standard, WLAN offers no guarantees for low communication delay. We then moved on describing the idea of mobile multi-hop ad-hoc networks that operate without any infrastructure components and can offer communication between wireless nodes even if both nodes cannot talk directly with each other. Because of interference and node mobility, routing is challenging in this environment. We concluded this section with an overview of existing approaches in this field.

In the second part of this chapter, we discussed distributed communication architectures and examined several client-server and peer-to-peer approaches. We specifically looked at related work concerning peer-to-peer systems for MANETs and existing architectures for multi-player games in the Internet. We concluded that no existing architecture is suitable to provide a reliable as well as scalable service for latency-sensitive applications in mobile multi-hop ad-hoc networks. As a result, we introduced our zone server architecture which combines client-server as well as peer-to-peer elements. To provide the necessary redundancy for applications in mobile ad-hoc networks, it selects a number of suitable clients as servers, which provide service for clients in their

vicinity. These zone servers are synchronized through a peer-to-peer network and can make authoritative decisions on behalf of the application as well as perform data aggregation. Our architecture uses zone servers as cluster heads thus exploiting any grouping of clients which we expect to see for social interactions with interactive applications in mobile networks. Furthermore, by aggregating data from their clients, zone servers can reduce bandwidth requirements and relieve long multi-hop connections. The server selection algorithm determines suitable clients to become zone servers and is described in detail in the following chapter.

4. The Server Selection Algorithm

In the previous chapter, we showed that the zone server architecture operates according to a divide and conquer principle. It splits up the network into multiple zones each having its own zone server serving a distinct group of clients. Finding suitable nodes that can act as zone servers among the group of mobile clients in the ad-hoc network is an important function of the architecture. It can be divided into two major tasks. First, a suitable initial set of zone servers has to be determined when nodes start their application. And secondly, this zone server system has to be maintained throughout the application's lifetime creating and removing zones and zone servers as required. In order to be applied in larger networks as well, these functions should be performed independently of the total number of nodes in the network, thus ensuring scalability.

In this chapter, we present the server selection algorithm as a fully distributed approach to select and maintain zone servers in mobile ad-hoc networks. We start by taking a look at the design goals for the algorithm. Then, we follow up with a detailed description of the algorithm and examine its three phases. To illustrate how server selection is done in a mobile ad-hoc network, we present an example. Furthermore we discuss the shortcomings of our approach and provide solutions for them. The subsequent section outlines the implementation of our algorithm in NS-2 which we will use as a simulation tool for our evaluation in Chapter 6. Finally, we conclude this chapter with related work and a summary. The initial version of our server selection algorithm was published in [147].

4.1. Design Goals

We have defined six objectives for our server selection algorithm which are the general principles for its design. In the following, we present and explain the reasons behind each goal:

1. **Discover other devices in the mobile ad-hoc network that are interested in using a particular application**
   In order to link nodes with similar interests together, mobile devices need to

know about other nodes in the network. Thus a discovery method is needed that allows mobile devices learn about other nodes running the same application in their vicinity.

2. **Select zone servers only among the group of nodes that run the same application**
   Nodes in a mobile ad-hoc network may run a variety of applications. The users of these nodes are usually perceptive regarding the lifetime of their device and the network bandwidth they are able to use because both factors influence the responsiveness of networked applications. For this reason, we believe that users are reluctant to sacrifice any network bandwidth, processing power or energy to act as server for an application that they are not running themselves. Other users that are running a specific application may be inclined to let their device take over a server role because it is of benefit to them.

3. **Select only nodes with sufficient resources as servers**
   We expect that mobile ad-hoc networks comprise a number of heterogeneous devices. Today, that variety ranges from small embedded devices to full-featured laptops including but not restricted to phones, game consoles and PDAs. Each of these devices has a unique set of processing and networking power, memory and battery lifetime. All of these attributes should be taken into account when choosing a suitable zone server. E.g. a mobile device which has insufficient battery lifetime or which lacks the processing power to function as server should not be considered as zone server. The basic principle behind this goal is the more powerful and the more durable a node is, the better the service we can expect from it.

4. **Prefer servers with a suitable position in the network**
   A good zone server has to be able to communicate efficiently with its clients as well as other servers in the network. Thus, a good server should have a suitable and well-connected position within the mobile network. We define such position by two characteristics. Firstly, in order to keep up with the requirements of latency-sensitive applications, the distance between a client and its server should be restricted to a few hops only whenever possible. By keeping server and clients close together, this behaviour also conserves precious network capacity in a mobile ad-hoc networks. Secondly, communication between zone servers should be as stable as possible as to avoid inconsistencies. Thus, a good zone server should preferably have links to multiple nodes to provide redundancy in case a radio link breaks. Because of the same reason, the selection of zone servers at the far edge of a network should be avoided.

5. **Constantly update neighbourhood information and adjust number of application servers as required**
   Due to the nature of the mobile ad-hoc network, changes due to mobility or interference may occur at any time. Thus the server selection algorithm should employ a mobility management mechanism that alleviates the effects of theses problems. Zone servers should be able to relinquish their server role and new servers should be created as the network changes. The selection of a new server should rely on local information only whenever possible without the need for additional inquiries if a server is lost. Therefore neighbourhood information should be gathered proactively.

6. **Server selection should work independent of the network size**
   Any algorithm in a mobile ad-hoc network needs to address communication costs and the limited energy of mobile devices to some extent. Therefore, we believe that a fully distributed approach which gathers information about the local neighbourhood of a node is preferable to a global and omniscient algorithm which gathers knowledge about the whole network. We believe that a distributed approach ensures scalability and is independent of the actual size of the network. Additionally and as stated in objective four, latency-sensitive applications prefer a close-by server. This means that only restricted local knowledge is necessary in order to determine a suitable zone server.

4.2. Algorithm Details

The server selection algorithm is a fully distributed algorithm that runs individually on every node. Each node regularly transmits information about its capabilities, status, and network connectivity to its neighbours and gathers information from others in a local neighbourhood table. The server selection algorithm then makes a local decision based on information in its neighbourhood table.

With the server selection algorithm, every node has three basic tasks to perform. Initially, mobile devices that run the same application or service have to be discovered in the mobile ad-hoc network. Then and based on this information, the algorithm selects an appropriate node to perform the functionality of a server. Finally, the algorithm performs periodic updates to adapt to changes in the network and account for the possibility that existing nodes disappear or new nodes are introduced. Accordingly to these three tasks, the server selection algorithm uses three different phases. During the *discovery phase*, initial information about the network and other nodes is gathered. In the following *selection phase*, nodes are determined as zone servers. During

the *maintenance phase*, network or node changes are detected and a new server selection process is initiated if required. Neighbourhood information concerning neighbour nodes and network links are periodically updated through all phases.

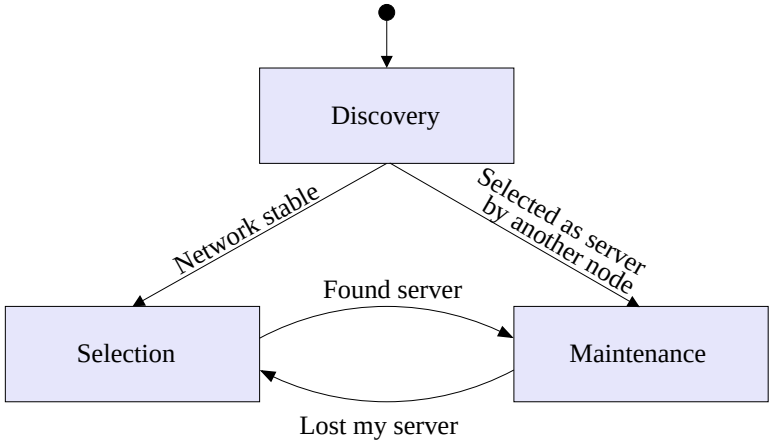


Figure 4.1.: Phases and transitions of the Server Selection Algorithm

Figure 4.1 shows all three phases of the server selection algorithm as well as the transition conditions between phases.The discovery phase mainly acts as an initialization period for the algorithm to ensure that neighbourhood information has a certain quality before making the first decision on which node becomes server. Therefore, the algorithm switches to the selection phase as soon as the network is considered stable. If a node joins the network while the server selection process is already in progress, it may happen that it is selected as server by another node. During selection, the algorithm looks through the local network table to find a suitable server in its vicinity. If a server is found, it is passed on to the application which can then make contact to its new server. If no server can be found, the selection process is repeated as new information is added to the neighbourhood table. During maintenance, the server selection algorithm remains dormant and waits for information from the application that a new server selection is required.

The server selection algorithm assumes that all nodes are in principle mobile although some or all of them may remain stationary for a long time. Thus, no special considerations are taken for infrastructure components that may or may not be present in the network. Such infrastructure components can be used, e.g., to provide long distance connections to remote mobile nodes, allow access to the Internet, or provide other services for the mobile ad-hoc network. As long as they are part of the mobile ad-hoc network, the server selection algorithm treats infrastructure nodes as any other node. Discussion on how infrastructure nodes can be used to increase stability and perfor-

mance of the network and our application is left for future work.

In the following sections we introduce the messages that are sent and received by the server selection algorithm and take a closer look at its three different phases.

4.2.1. Announcement & Selection Messages

Before the server selection algorithm can start choosing a suitable server for its application, it needs to know which nodes are available in the vicinity and if they are capable of acting as server for the network in addition to their own responsibilities of running the client application for their local users. Therefore, each node determines its performance capabilities which may include CPU performance, memory and storage capacities, energy consumption & supply as well as its available methods of communication. All this local information is compressed into a single metric which we call the weight of a node. The greater its weight, the more capable a node is. In the case where a node deems itself incapable of acting as a zone server, it uses a weight of zero as a special value. Nodes with a weight of zero are never chosen as zone servers. The individual composition of the node's weight is left to the application. A more detailed discussion about which resource should be utilized to define a node's weight can be found in [148].

Additionally, the position of the node is also important for a prospective zone server. Hence, we use the number of links a node has to other neighbours as additional metric for the server selection algorithm. We define the node's degree as the number of edges a node shares with other nodes in a connection graph that shows direct single-hop communication. Nodes with better connectivity in the network are favoured when selecting zone servers. A more central position in the network allows prospective zone servers to reach more nodes within the time requirements of a latency-sensitive application. A larger number of links may also provide a more stable services because a greater number of routing alternatives can be found in case one or more nodes move.

The node's weight and degree are used as performance and network metric respectively. They are both part of periodic server selection announcements that are broadcasted regularly by every node running the server selection algorithm. Each node manages its own local neighbor table in which it stores information received from announcements messages of other nodes. Usually, the number of neighbouring nodes can be estimated by looking at information from the lower layers, e.g. from the ARP or routing tables or from periodic link layer announcements. Furthermore, a node may also rely on the number of entries in its own local server selection neighbourhood table to determine its own degree. The concurrent use of multiple sources in determining

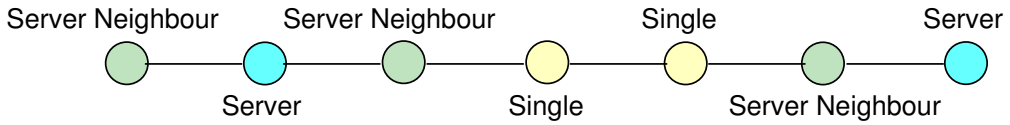


Figure 4.2.: The different roles of a node

the number of neighbours is advantageous because it can speed up the initial setup time until stable neighbour information is transmitted, therefore reducing the convergence time of the server selection algorithm. In [149], we proposed to piggyback server selection announcements onto existing WLAN MAC layer beacons to improve neighbour detection with negligible overhead.

Besides a node's weight and degree, an additional status flag is present in server selection announcements. It describes the one of three different states or roles that a node can take in the network. The role of a 'Server' indicates that this node was chosen as a zone server for the application and has accepted this role. A 'Server Neighbour' is a node that is a direct neighbour to a zone server. As soon as a node announces its status as a zone server, all adjacent nodes that are not a server themselves switch their roles to 'Server Neighbours'. The third status is the role of a 'Single' node. Single nodes have no known direct link to a server and are not zone servers themselves. Also, the 'Single' role is the initial status of a node when the server selection algorithm starts. Having the role of a single node does not necessarily mean that the node has no information about nearby zone servers. Figure 4.2 shows an example of a mobile ad-hoc network after the server selection algorithm has determined suitable servers. It shows two server nodes as well as their server neighbours and two single nodes. Irregardless of their role, all nodes can determine a suitable server for their application in this scenario. For server and server neighbour nodes, the solution is obvious. Single nodes on the other hand can make contact to a zone server simply by going through their adjacent server neighbour nodes. Thus by having a node with server neighbour status as neighbour, any node can conclude that it can reach a zone server within two hops.

Finally and to allow multiple server selection processes working concurrently in a mobile ad-hoc network, we use a unique application ID from which we can distinguish different applications running concurrently in the network. While the exact definition of such a unique ID is beyond the scope of this thesis, several different approaches are conceivable. Similar to the assignment of MAC addresses, each software vendor could be assigned a globally unique vendor ID. The vendor could then allocate a local product ID to each of his software products. The combination of vendor and product ID would then provide again a globally unique application ID. Other possibilities include

Field	Purpose
Message type	Used to distinguishes two kinds of messages: – Announcment message – Selection message
Node weight	Metric of node's resources (Performance metric) (0 = unable for server role)
Node degree	Number of known direct neighbours (Network metric)
Node state	The current state the node is in: – SERVER – Node is server – SERVER NEIGHBOUR – Node is direct neighbour of a server – SINGLE – Node has no adjacent server
UAID	Unique application ID which identifies the application which carries out the server selection

Table 4.1.: The announcement message of the server selection algorithm

a hierarchical naming scheme similar to the domain name system (DNS)[48, 49] or to the service tree in Konark[150], a service discovery protocol for mobile ad-hoc networks.

Both metrics, the node's state and the unique application ID are part of the announcement messages that are sent periodically by every node that participates in the server selection process. Table 4.1 gives an overview of all items in the announcement. The actual interval between two consecutive transmissions of a server selection announcement message of a single node depends on the status of the server selection algorithm and will be discussed on the following pages. Generally, when using contention based medium access protocols such as IEEE 802.11 (WLAN), it is advisable to randomly reduce or increase the waiting period between two consecutive transmissions of periodic messages to avoid collisions. For this reason the server selection algorithm varies its sending interval randomly in order to reduce the possibility of concurrent transmissions of multiple nodes. Announcement messages are always broadcasted locally and are never forwarded by a receiving node. As can be seen from Table 4.1, our algorithm also uses a second so called selection message. This message is used by any node after its local server selection algorithm has determined a suitable server candidate to inform the server candidate node that is has been chosen by the sender of the selection message. A selection message contains only a single field, the message type. We will discuss selection messages as part of the selection phase later on and now take a look at all three phases in detail.

4.2.2. Discovery Phase

The discovery phase is the first phase of the server selection algorithm. It is responsible for creating an initial and stable view on a node's neighbourhood on which the first server selection decision can be based. Whenever an announcement message is received from another node, the local neighbourhood table is updated. This table contains all information from the announcement message which is shown in Table 4.1 as well as the neighbour's network address and a local timestamp indicating the last time an update was received from that particular neighbour node. This timestamp is used to remove nodes from the neighbourhood table if no announcement was received for a longer period of time which usually indicates that the node has left the network neighbourhood. The neighbourhood table also contains information about the local node which is updated regularly as it sends its announcement message. This procedure ensures that the local node is considered in the server selection process as well. We will show an example of a neighbourhood table later on in Section 4.2.5.

We have shown in Figure 4.1 that the transition to the selection phase occurs when the network is considered stable by our algorithm to perform the first server selection. To achieve this condition, at least *min_tx* announcement messages must be sent by the local node. This initial delay ensures that messages from all neighbours are received before making a server decision. Because of the randomization of the announcement sending interval at each node, *min_tx* should have a value greater than two messages. As the value of *min_tx* defines the lower time bound for the convergence of the server selection algorithm, careful considerations should be taken into account to find a good balance between network stability and fast convergence. When the *min_tx* messages are sent, the server selection algorithm determines if it's neighbourhood table has stable information by looking at two consecutive updates from each neighbour. As soon as updates from neighbours do not change the neighbour node's weight or degree, the network is considered stable. To prevent the algorithm from staying in the discovery phase forever if the network changes constantly, *max_tx* defines the maximum number of announcement messages the local node can sent before the algorithm forces a transition to the next phase.

4.2.3. Selection Phase

During selection phase, the node chooses its best server based on data from its neighbourhood table. First, neighbours that are unsuitable for taking the server role are excluded from the decision process. These are nodes that have a weight of zero and are therefore deemed inappropriate servers due to the lack of local resources. Further-

more, nodes with a degree smaller than two are also not considered because of their unsuitable or unknown position in the network. Nodes with a degree of one are leaf nodes to which the local node has the only link. Nodes with a degree of zero were obviously just switched on or moved in from another location. As long as they have incomplete information about their position in the network, they are disregarded.

For the remaining node in the neighbourhood table, a server is selected according to the following three rules:

1. The node with the highest weight is considered first.
2. If two or more neighbours have the same weight, the one with the highest degree is chosen.
3. If two or more neighbours have the same weight and the same degree, the one with the highest network address is chosen.

If the node does not have a neighbour yet or is not able to find a suitable node, it delays server selection until its neighbourhood table changes. If the node itself is the most powerful or best connected node in the neighbourhood, the server selection algorithm may choose the local node to operate as zone server. It then changes its status to server and broadcasts its new role to the network through an announcement message. However, additional checks are being done before a node selects itself as the best zone server candidate. If there is another zone server nearby with the same weight and the same degree as the local node, it refrains from choosing itself but connects to the existing server instead. In other words, the local node prefers an existing server if it is as capable and well-connected as the local node itself. Such situations can happen because the server selection algorithm runs locally on every node and therefore converges faster on one node and slower on others. Also, a local server does not select itself if it has recently dropped its role as a server. This behaviour prevents oscillations where a local server switches between client and server role frequently due to a changing network environment.

If a node selects another node as its most-suited neighbour for becoming a zone server, it informs this node by sending a selection message. Aside from the sender's address, this message contains no other information than that a node has been selected by another node as its server. From the number of received selection messages, the prospective server can get an estimate of how many clients it can expect later on. Selection messages are unreliable and of informal nature only. They serve as an out-of-band signaling method for the server selection algorithm to inform nodes and their application about their new prospective server role. If a selection message gets lost during transmission, the local node can detect that its chosen server not has not yet switched its role to server. Thus, it will re-run the selection process at a later time and the se-

lection message is automatically retransmitted. Depending on the application, it may also be feasible to inform the prospective server of its chosen role by simply telling the application to connect to the new server. Both notification approaches may be used, even simultaneously. Upon reception of a selection message or application data from a client, the node changes its role to server and again informs its neighbourhood through an announcement message.

If a node becomes a zone server, all neighbouring nodes get notified through the server's announcement messages. In cases where a node does not receive the update because of interference, it still gets the information eventually through one of the next announcement cycles. When a node recognizes that it is now the neighbour of a zone server, it changes its role to server neighbour. If no suitable node can be chosen as zone server, the server selection algorithm tries to find an existing server nearby. In the case that the local node is a server neighbour itself, it can simply connect to one of its neighbouring servers. If the local node is a single node, it has no direct connection to a server. It then looks through its neighbourhood table to find a server neighbour through which it is guaranteed to reach a server within two hops. If everything else fails, the node has no server nearby and remains in its role as a single node. However, it continues to find a suitable server as the network changes. If the local node remains in this status for a prolonged period of time, it may revert to alternative server selection methods to find a suitable server, e.g. by using restricted flooding.

If a suitable server has been chosen by a node and if the server has accepted its server role by propagating it in its announcement message, the server selection algorithm switches to the maintenance phase. In case no server can be found in the vicinity, the node remains in selection phase. Then the selection process is repeated periodically every time an announcement message is being sent.

Figure 4.3 shows a simplified version of the server selection algorithm. The red and green boxes represent a role change of the local node while its colours indicates whether the local node knows a server to which it can connect. For the sake of simplicity, we used two separate threads to differentiate between the transmission of messages and the server selection process on one hand and the message reception on the other hand. In reality, the reception of packets can be done as the first thread waits for the announcement interval. Thus, the reception of a selection message and the subsequent role change into a zone server prevents the algorithm from executing another server selection round and leads directly to the maintenance phase.

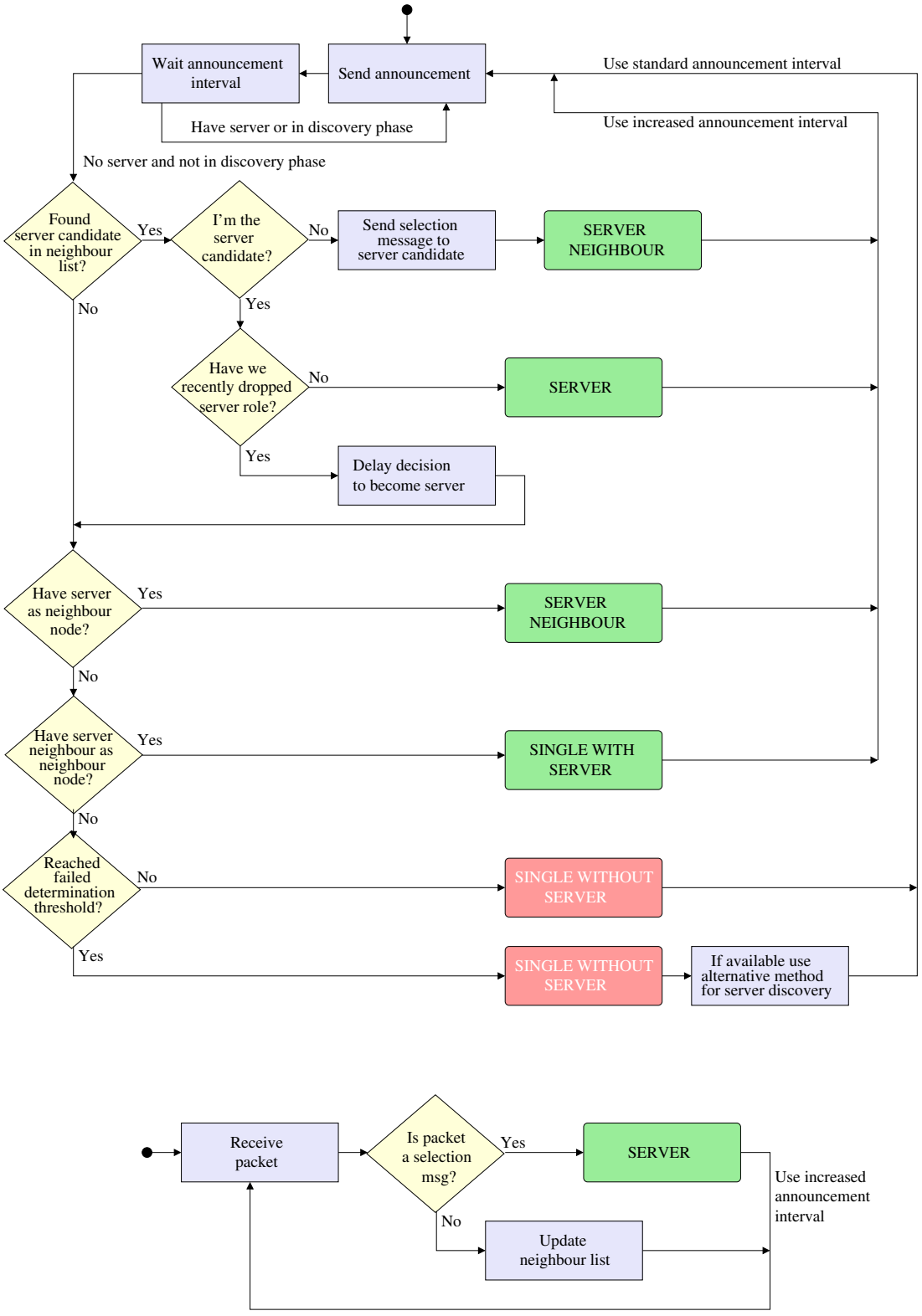


Figure 4.3.: Flow chart of the server selection algorithm

4.2.4. Maintenance Phase

When the algorithm transitions into maintenance phase, the node has successfully selected or discovered a server with which its application can communicate. As long as the network does not change as nodes move, get switched off or network connections are impaired due to outside interference, the connection to the server remains stable and the server selection algorithm stays in maintenance mode. Here, it continues to send and receive server selection announcement messages to detect network and node changes but otherwise stays dormant. As indicated in Figure 4.3 the server selection algorithm uses an increased period for announcement to conserve network bandwidth. Still, announcement messages need to be sent periodically even in maintenance phase to allow new nodes to discover their neighbourhood.

An important feature of algorithms for mobile ad-hoc network is their mobility management meaning how they deal when nodes start moving around. In our case, mobility can affect the connection between client and server in four different severities. Firstly, small movements generally go unnoticed by the application. They may lead to slight variations in link quality between neighbours as the signal strength of received packets changes. Secondly as the signal strength decreases further, packets may get lost during transmission. Here, the underlaying link layer protocol may decide to switch to a slower but more robust transmission speed which decreases performance but increases the chance that a packet can be received successfully by its recipient. Consequently, the application can notice an increased number of lost packets, an increased packet delay as packets get automatically retransmitted, and/or a greater variation of latency. Furthermore, as a node decreases its transmission speed, it increases its utilization time of the wireless medium which can influence traffic of neighbouring nodes as well. If a node moves further, eventually network links will break. Now, the MANET routing protocol becomes active trying to either repair broken routes or replace them with new ones. During this time, the application is unable to send and most likely receive packets from nodes beyond the broken link. Thus, route changes for the application often mean an intermediate but complete loss of communication with some or all nodes in the network. As data packets are usually buffered, they experience delays that can reach up to multiple seconds, sometimes tens of seconds before the queued packets can be successively transmitted through a new route. Finally, routes may fail completely when a network link breaks. This happens if there is no alternative route available and the network is split. An application notices a loss of communication often only after some tens of seconds when buffered packets are deleted from the network queue and an error packet is generated. In MANETs, communication may be lost for an undetermined period of time if nodes move.

In maintenance phase after the server selection algorithm has discovered a server, it is up to the application to decide whether its connection to the server is satisfactory. If communication degrades up to a point where the application determines that a transition to a better server is advised, it reports back to the server selection algorithm. The algorithm then switches back into selection phase and tries to find or to create another suitable zone server. By transmitting and receiving announcement messages to and from other nodes even in maintenance mode, the server selection algorithms ensures up-to-date information about the node's network neighbourhood at any time. Thus, the algorithm is able to make a fast decision on a new server as long as a suitable candidate is nearby. For clients, mobility can influence communication to their respective servers in several different ways. If a link breaks and the routing protocol is able to find an alternative, communication between client and server can be restored. However, the new route may now include an increased number of hops often meaning an overall degraded network performance for the client. By design and to offer good network performance, the server selection algorithm only discovers servers that are at a maximum of two hops away from the client node. After a route change, a new route to the server now may have a distance of more than two hops. However, the two hop limit is not enforced after an application makes contact with its server. Regardless of client-server distance, it can continue to use a particular server as long as it believes the network performance is satisfactory. In any case, an application is free to report the loss of its server or any service degradation to the server selection algorithm at any time. By moving this responsibility to the application, we can ensure that this decision can be tailored to its individual demands. Such behaviour also allows for a parallel and proactive search for a new server even if the connection quality to the actual server is still within the necessary limits.

For a server, the loss of a nearby link can affect multiple clients at once. The local user is only indirectly affected because his application is connected to the local server instance. As the server selection algorithm operates in a client-orientated fashion which lets the client decide which server to use, it has little to do in that regard. However, there are various other tasks for the server during the maintenance phase most of which concern the application. First of all, changes to the network can have an impact on synchronisation between servers as well. Also as nodes move around, the server selection algorithm chooses additional nodes as new zone servers for the application. Therefore, unused servers should relinquish their server role and revert back to simple clients. The server selection algorithm allows the application to decide when the server role should be dropped. Normally, such a role change should only be done when the server has had no more external clients for a certain period of time. If that happens, the local node automatically looses its local server and the server selection algorithm consequently switches back to selection phase. As a previous server, the application may also have

additional information about other servers with which it synchronised application data before which could assist in finding a suitable server. However, such consideration is application-dependant and left for future work. Zone servers that revert back to client role are prohibited from taking over the role of a zone server for some time. Also, if new zone servers are chosen by the server selection algorithm, they are allowed a grace period during which they will not drop their server role even if they have no clients. Both provisions were created to avoid oscillations and to take into account that the creation of a new zone server requires additional network and processing resources in order to find other zone servers and setup a synchronization scheme.

A full list of parameters for the server selection algorithm can be found in Appendix A.1.

4.2.5. Example

We now give an example of how the server selection algorithm works in every phase. Figure 4.4 shows mobile nodes and network links of a mobile ad-hoc networks. We generally differentiate between two kinds of nodes. First of all application nodes that run an application that is using the server selection algorithm. Secondly, nodes that do not run the server selection algorithm and therefore do not take part in the decision process. Like with every nodes in a mobile ad-hoc network, they act as routers and forward unicast data to receivers throughout the network. Therefore, we call these nodes supporting nodes. In order to keep our example simple, we use a single application only although the server selection algorithm supports different applications running simultaneously. Nodes that run the server selection algorithm send out periodic announcement messages to their direct neighbours. Supporting nodes do not transmit any announcements messages themselves nor do they forward any announcement messages from other nodes. Supporting nodes can therefore be seen as obstacles for the server selection algorithm as they do not provide any information about their status to the network around them. Figure 4.4 also shows the weight of each application node. We can see that all nodes but node G have sufficient performance and resources to take the server role.

Every application node maintains its neighbourhood table which is used to find a suitable server. Table 4.2 shows an example of such table for node D which has a central position in the network. We decided to show a sorted neighbourhood table in accordance with our server selection algorithm so that the best node is always shown at the top of the table. In this case it is node B because it has the highest weight. Also, node B shows a degree of three, meaning that it has links to three other nodes. As supporting nodes are not part of the server selection process, they are not discovered through announcement messages. Nodes have to rely on other means, e.g. their routing table,

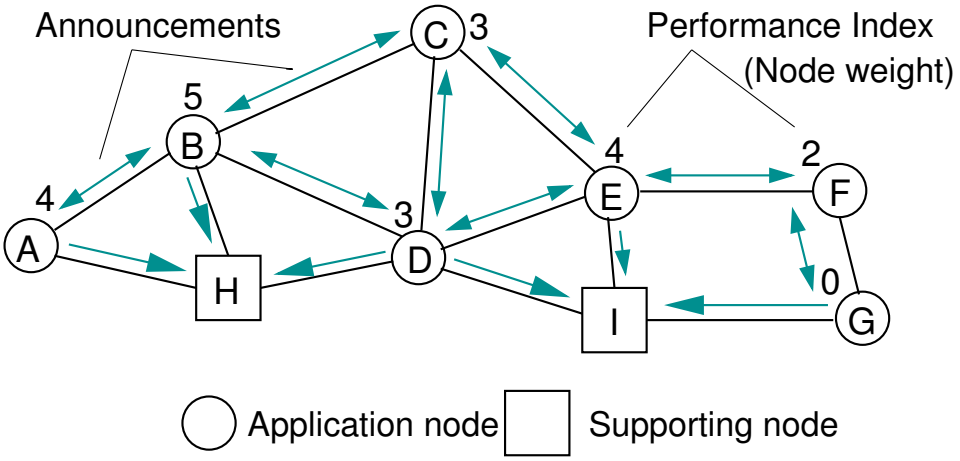


Figure 4.4.: Discovery Phase

Node	Weight	Degree	State
B	5	3	Single
E	4	4	Single
D	3	5	Single
C	3	3	Single

Table 4.2.: Neighbor tables of nodes D in discovery phase

to detect supporting nodes. To show that this method is not always reliable, we assume that node B has no knowledge about its link with supporting node H, while all other nodes have been able to gather that information. In any case, neighbours that are application nodes are always discovered through their announcement messages. Initially all nodes are single nodes, because we are still in the initial discovery phase and no selection process has been done yet.

After the neighbourhood information has been stabilized or the time limit for the discovery phase has been reached, the server selection algorithm switches to the selection phase. As each application node makes that switch for itself, this phase transition does not happen simultaneously for all nodes. Hence, a mixture of nodes in discovery and selection phase may exist for a certain amount of time. This, however, does not constitute a problem because nodes that are still in discovery phase just select their servers a bit later. To keep our example simple, we assume that all nodes are now in selection phase. All application nodes will look through their neighbourhood table to find the most suitable server for themselves. If we return to the neighbourhood table of node D in Table 4.2, we can see that node B is the best zone server candidate. Node D also

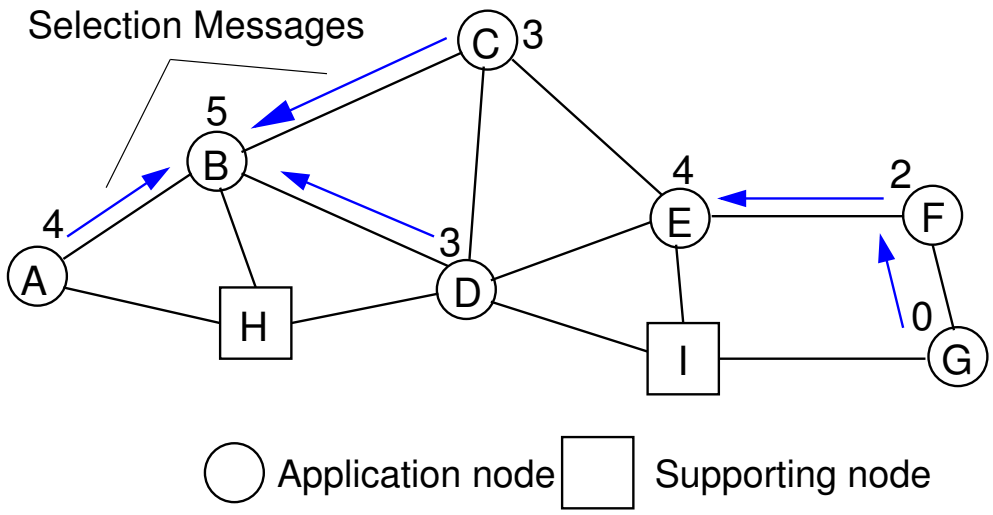


Figure 4.5.: Selection Phase

verifies that its selected server candidate has a weight greater than zero which indicates that B is generally able to take over server responsibilities. It also checks that B has a degree greater than one to avoid that a leaf node is selected as server. As all checks are passed, node D sends a selection message to node B and informs its application that a server has been found. Figure 4.5 shows the messages that are sent during selection phase. Table 4.3(a) again shows the neighbourhood table of node D after all nodes have finished their selection process and adjusted their roles accordingly. We can also see that node D is neighbour of two zone servers because of its central position in the network. Hence, should node B fail for any reason, D would be able to switch over to server E without any additional delay.

Let's take a look at the right side of the network. Here, node F has only one neighbour, node E, which is capable of performing the task of a zone server. Furthermore, node E has also a greater weight than F itself, so F chooses node E as its server. Also, node E is the most powerful node in its neigbourhood so it chooses itself. Finally, node G has little choice because it knows only one node being capable of performing as zone server, so it sends its selection message to node F. The resulting network is shown in Figure 4.6. It shows the area of influence for each zone server which basically covers all nodes that would select a particular zone server as their server. As server selection is client-driven with our algorithm, this zone is defined by the nodes themselves rather than the zone sever. We can also see that the initial server selection process overdetermined the number of zone servers in this example because server E has no other clients besides itself after node F has become a zone server. This is because only local information is available at each node which in this case is inadequate to find a suitable

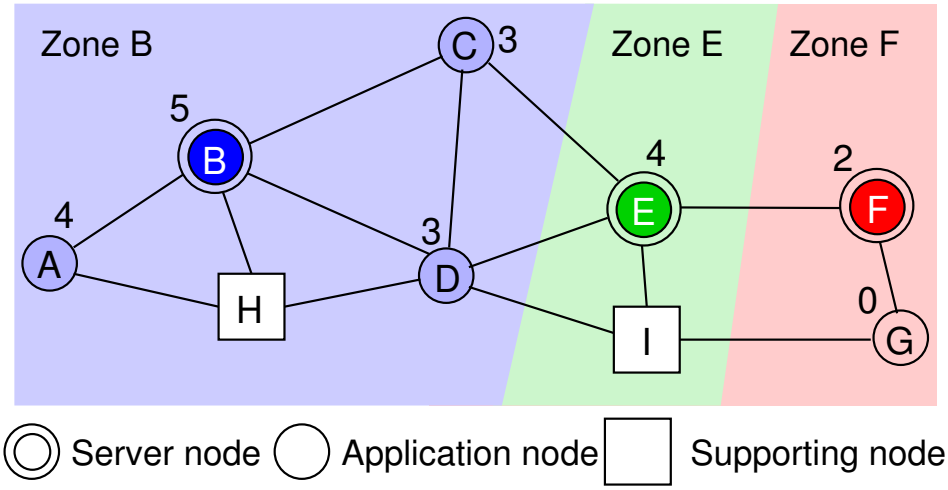


Figure 4.6.: Network after first server selection

(a) Node D

Node	Weight	Degree	State
B	5	3	Server
E	4	4	Server
C	3	3	Server Neighbour
D	3	5	Server Neighbour

(b) Node F

Node	Weight	Degree	State
E	4	4	Server
F	2	2	Server
G	0	2	Single

Table 4.3.: Neighbour tables of nodes D and G after server selection

solution during the first selection process. Thus, we need an additional adjustment that will happen automatically later on during the maintenance phase. Table 4.3 shows the neighbourhood tables for nodes D and F after the first selection phase. Here we can see that every node which received a selection message has accepted its server role and subsequently switched its state to server. They propagate their new role through an announcement message which prompts the state change of their neighbours to the role of server neighbours.

After the selection process is finished, every application node in our example has selected its zone server to which the application can connect. The server selection algorithm now switches into maintenance mode in which each node monitors changes in its neighbourhood. During this phase, server E notices that no remote client has tried to make contact. This is because node F which was the only node to send a selection

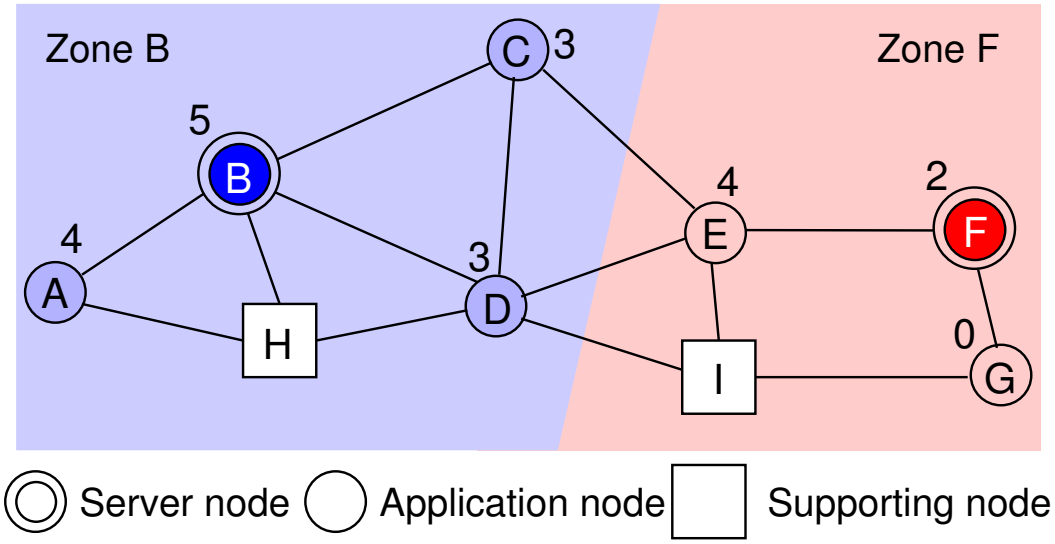


Figure 4.7.: Network after stabilization

message to E has become a zone server itself and now handles its own application locally. Consequently with no other clients than its own local application server E will eventually drop its server role. With this action, the local application on node E will automatically loose its server which prompts the server selection algorithm to do another server selection. By looking at its neighbourhood table, node E determines that it is still the best server candidate in its neighbourhood. However, because it dropped its server role recently, E is prevented from taking back this role for some time. Instead, it continues with the server selection process and tries to find another suitable server in its vicinity (see the selection algorithm in Figure 4.3). Server F is the only other server in E's neighbourhood table and is therefore chosen as the new zone server for node E. The actual server selection process can be done without any delay because E already has all the required information in its neighbourhood table. After its application made contact with its new server, E's server selection algorithm switches back to maintenance phase. The network has now reached its final stable state which is shown in Figure 4.7. As long as all network links remain active and no nodes are being removed from the network, the server selection algorithm at each node remains dormant in maintenance mode.

If a node starts to move around, it looses some of its old network links and creates new ones. Figure 4.8 continues our previous example as node G starts to move away from its zone server. Initially and as long as the application can deal with additional delay and packet loss that comes with the creation of new, most likely longer routes to the server, node G maintains its connection to server F. It continues to send and receive

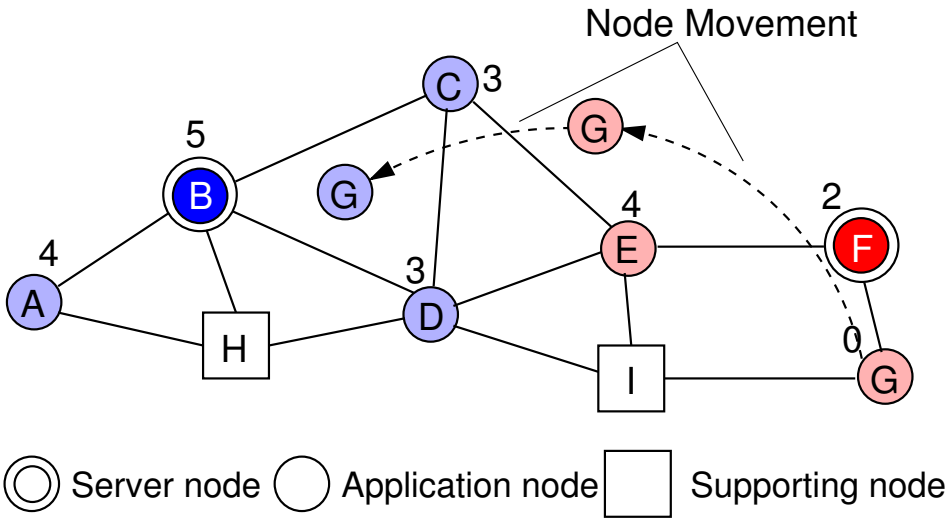


Figure 4.8.: Mobility

server selection announcement messages. On its way through the network, node G first discovers node E and then later on nodes C, D, and B. As the server connection starts deteriorating further and the mobile node moves further away, the server selection algorithm starts looking for a new server. At its final position, it can connect to node B as its new server. If the application requested a server change earlier, a new server would have been created. Because of the path that node G takes through the network, this server would most likely be node E.

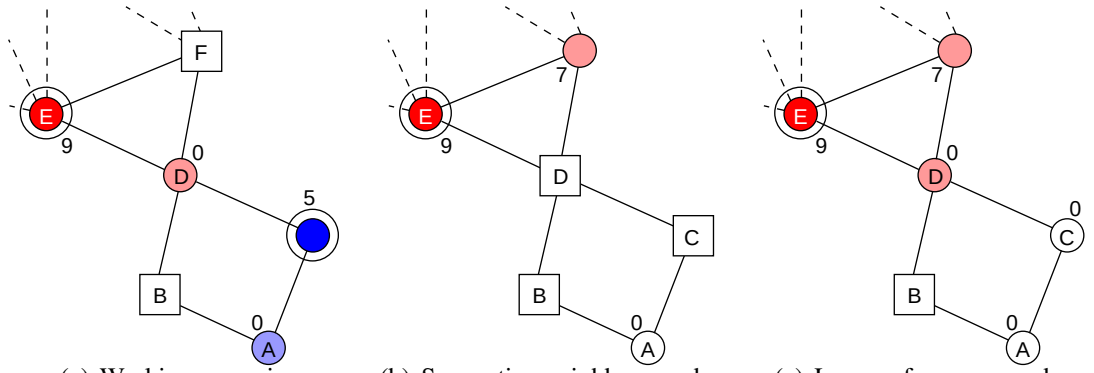
4.2.6. Discussion

There are some limitations of the server selection algorithm that are noteworthy and which we will discuss below. Depending on the composition of the mobile ad-hoc network and the existence of supporting nodes, the server selection algorithm may not be able to select a suitable server in every conceivable scenario. This is due to the fact that not every node in the network can take the role of a zone server, e.g. supporting nodes or application nodes with a weight of zero. Figure 4.9 shows three different situations of the same part of a mobile ad-hoc network.

In the left scenario in Figure 4.9(a), all application nodes have at least one link to another application node. Furthermore, both low performance nodes A and D are distributed so that suitable zone servers can be chosen by the server selection algorithm. In Figure 4.9(b), we replace nodes C and D with supporting nodes. As a result, node A is now isolated in the network and has no chance of sending or receiving any an-

nouncement messages to/from other application nodes. Although the server selection algorithm works for other parts of the network, it fails to select a server for node A. In these situations existing alternative server or service discovery mechanism can be used as a fallback solution. One example of such a fallback algorithm is restricted flooding where A sends a broadcast message in order to discover other application nodes or servers. The difference between restricted flooding and our announcement messages is that restricted flooding uses a maximum hop-count that is greater than one. As such, discovery packets are forwarded by other nodes including supporting nodes which they do not do for in the zone server architecture. Another possibility is to rely on existing service discovery protocols to assist standalone and remote nodes to find a server. The Service Location Protocol (SLP)[151] uses broadcasts or an SLP directory server to locate services in a network. While SLP was designed for local area networks such as ethernet, several modified versions exist that provide service discovery in MANETs. Another alternative is Konark[150], a protocol especially designed for service discovery in MANETs. The third scenario in Figure 4.9(c) shows a situation where all nodes have at least one application node as neighbour. However, server selection is impaired as three nodes in a row do not have sufficient resources to become a zone server. For nodes D and C the server selection algorithm is able to find a suitable server. Node D has a direct connection to server E and node C is able to discover its server through server neighbour D. For node A, however, the situation looks different, because its only neighbour node is node C which has single status. The server selection algorithm fails in this case because of its limitation by design that zone servers are only selected at a maximum of two hops in order to ensure a good network performance for the client. In this situation node A could directly enquire if node C has a server and ask for its address. Of course, the previously discussed server discovery alternatives could also be applied here. In Chapter 5, we will show through simulation that the limit of 150 ms delay and 5 % packet loss in a MANET can normally be achieved at a maximum of three hops. By limiting server selection to two hops, we create a more stable network setup that does not operate too close to the applications performance limits thus allowing a satisfactory user experience even in varying network conditions. Furthermore, if the number of hops between server and client increases due to node mobility, the server selection algorithm still has enough time to find another suitable server before the connection to the original server deteriorates further.

The server selection algorithm is a fully distributed method which does not have any global knowledge. This approach is scalable and can support mobility without any unreasonable overhead. But as such it is unable to fulfill global constraints like to enforce a minimum number of zone servers for the entire mobile ad-hoc network. However, from an application point of view redundancy is essential to keep up its service despite of failing nodes. To solve this problem, we propose that the application



(a) Working scenario (b) Supporting neighbour nodes (c) Low performance nodes

Figure 4.9.: Impact of supporting and low performance nodes on server selection

takes care of enforcing global constraints as it sees fit. As all clients are connected to their respective zone servers which are in turn interconnected for synchronisation purposes, the application has sufficient information to make such decisions. It also can decide for itself how much server redundancy is required. If any application wishes to create a new server or shut down an existing one, it simply needs to notify the server selection algorithm that the role of a node has changed. If the algorithm is able to publish a node's new status in its announcement messages, it will act automatically to rectify the situation if required.

Finally, additional information from the application may be used to assist the server selection algorithm to speed up server discovery if nodes move around. E.g. if a global server list is available at the application level it may be used to supplement existing information in the neighbourhood table. However, we leave advanced interaction between the server selection algorithm and the application for future work.

4.3. Implementation

We have implemented the server selection algorithm in C++ as an application protocol in NS-2[152]. Figure 4.10 gives an overview of our implementation. It includes the server selection algorithm which is split up in two major modules. The selector module encompasses the algorithm itself including message handling and phase transitions. The node table module implements the neighbourhood table and calculates the best node to be used as a zone server candidate. Additionally and for the purpose of evaluating our approach, we implemented an example application which uses a traffic model that is based on the fast-paced multi-player game CounterStrike. The interfaces between all modules are kept very small and basically contain only a few calls in each

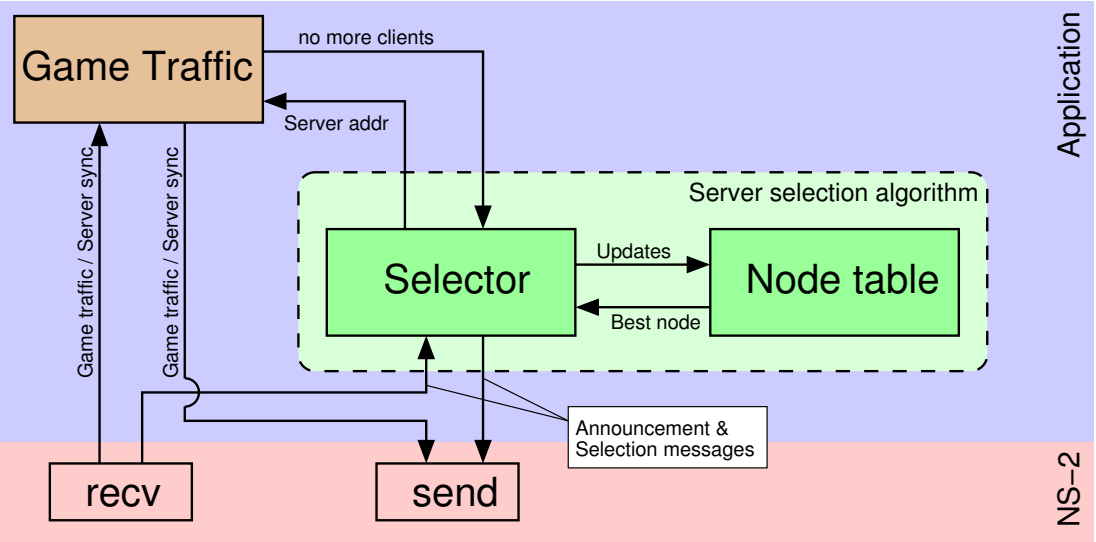


Figure 4.10.: Implementation details

direction. Additionally, our implementation uses only two NS-2 primitives to send and receive data from the network. Hence, little effort is needed to compile our implementation for existing mobile hardware. In the following paragraphs, we take a closer look at each module.

The selector module is responsible for maintaining the algorithm's state machine as well as collect information for and send out regular announcement messages. To avoid that multiple nodes repeatedly send their announcement messages at the same time, the selector module diversifies the sending interval by $\pm 20\,\%$ which should be sufficient even for larger networks. Figure 4.11 shows the announcement message format for IPv4. Every timer and threshold of the server selection algorithm is parameterized and can be set individually for each node through an NS-2 scenario file writte in TCL. As the selector module receives announcement messages from other neighbouring nodes, it forwards this information to the node table module which updates the neighbourhood table. Also any changes in weight or degree for the local node is passed on to the node table module which needs to keep current information about the local node in case it needs to select itself as zone server. During selection phase and as shown in Figure 4.3, the selector module requests the best server candidate from the node table module and runs the selection algorithm. The selection module offers various parameters which can be set by the user. One example is the weight of each individual node. The degree of a node is automatically calculated by adding up all remote nodes from the neighbourhood table. If a server is selected, the selector module forwards the server's address to the game traffic module. The node table module implements the

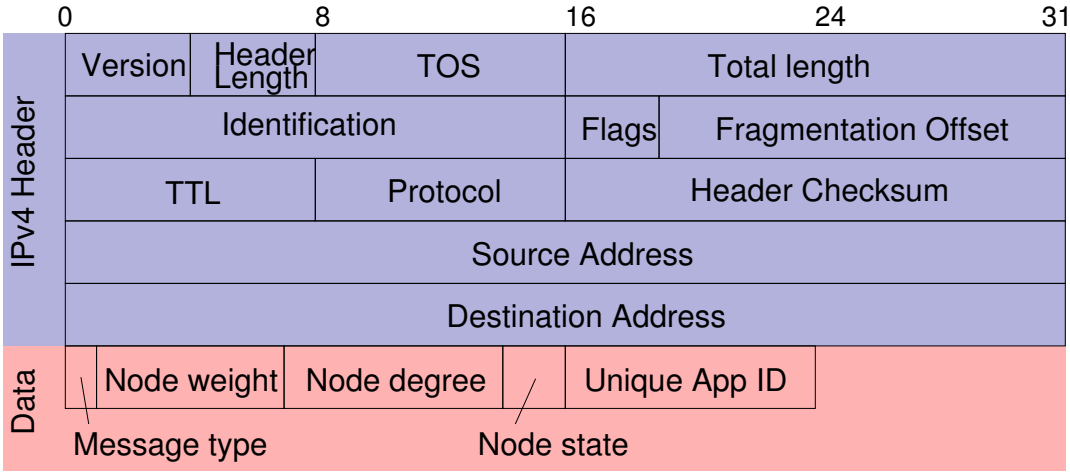


Figure 4.11.: The announcement message format for IPv4

neighbourhood table and contains methods to fetch and update entries. Furthermore, the server selection rules as described on page 93 are implemented here.

The game traffic module is not part of the server selection algorithm. It servers as an example application which we will use in our evaluation later on. It is able to reproduce the traffic pattern of a latency-sensitive application and can operate in client as well as server mode. We chose the multi-player game CounterStrike as our example application because it is a popular, fast-paced and highly interactive application. Also, the analysis of the network traffic for CounterStrike has been the goal of various research papers[8, 53, 153] and is particularly well-understood. We will discuss its traffic characteristics in more detail in the evaluation in Chapter 6. The game traffic module stays dormant until it is being notified by the selector module that a server has been discovered. If the address of the server matches the local address, the game traffic module changes to server mode otherwise it switches to client mode.

In client mode, the game traffic module behaves like a normal CounterStrike client and sends and receives unicast UDP packets to and from its server. With our application, a packet from a client represents an action that was taken by the user. The server collects all user actions and periodically creates a new game state which it propagates back to its clients. For evaluation purposes, the client numbers every sent packet and records the delay between an user action and its corresponding game state update as responses from the server come in. If a client looses the connection to its server, it stops transmitting action packets and informs the selector module which then tries to find a new server.

In server mode, the game traffic module receives action packets from its clients and

sends game state updates back to them. Each game state update includes all action numbers that were considered for this update so that any packet loss can be detected and delay can be measured by the clients. However, our module still ensures that game state update packets resemble original traffic characteristics for a CounterStrike server. Each server keeps an up-to-date number of clients it serves in order to be able to decide when it has no more clients and should drop its server role. The game traffic module also manages the grace time period that allows new servers to keep their role for some time even if they currently have no clients. Because there is no mechanism that tells the server that it has lost a client, the server makes that determination by using a individual timeout counter for each client. In addition to client traffic, servers exchange additional messages between each other to synchronise their game states. As CounterStrike is originally an application with a single server, there is no data how such synchronisation traffic would look like. Therefore, we decided that servers exchange the same game state update packets they sent to their clients. If a server drops its role, all transmissions are stopped and the selection module is notified.

In Appendix A.1 you can find a complete list of parameters of the server selection algorithm together with their default values.

4.4. Related Work

Determining a group of servers where each server is in the vicinity of a group of nodes is similar to a well-known graph theory problem of finding a dominating set. A dominating set describes a subset of nodes of a graph and is defined as follows: For all nodes, either the node itself or a direct neighbour is a member of the dominating set. A dominating set that uses the minimum number of nodes is called minimum dominating set. Determining a minimum dominating set (MDS) is an np-hard problem.

Kuhn and Wattenhofer[154] introduced a distributed algorithm to find a fast and non-trivial approximative solution to the MDS problem in a constant number of rounds. They achieve this goal by using information about the degree of all neighbours up to the distance of two and a suitable weight function. CEDAR[98] is a distributed routing algorithm which supports quality of service. It relies on an approximated MDS to calculate a core of nodes which are used to forward traffic. In [99], Wu proposes a routing scheme which uses a dominating set algorithm to determine a backbone in mobile ad-hoc networks. Traffic is routed only by nodes in this backbone while other nodes simply transmit their traffic to the backbone nodes. While similar characteristics between our algorithm and these approaches exists, the server selection algorithm does not determine a dominating set. Instead we allow game clients to be up to two hops

away. Also, we take the heterogeneity of mobile ad-hoc networks into account and distinguish different device capabilities as well as their user's intention of becoming a server in addition to their position in the network.

In 2006, Károly Farkas published the Priority Based Selection (PBS) algorithm[155]. Similarly to our server selection algorithm, PBS determines suitable servers in a mobile ad-hoc network for zone server-based applications. It is part of SIRAMON (Service provIsioning fRAMework for self-Organized Networks) which aims at combining common services for applications in mobile ad-hoc networks into a middleware. SIRAMON offers service description & discovery, deployment as well as service management. Because of a cooperation between Prof. Plattner, ETH Zürich and Prof. Wolf, TU Braunschweig, PBS and our server selection algorithm were not developed independently from each other. Therefore, both approaches exhibit some similarities with regard to the algorithm itself, the used metric and evaluation scenarios. We will therefore discuss the priority based selection algorithm in the following paragraphs and point out the differences to our approach.

The priority based selection algorithm calculates a dominating set of nodes in a mobile ad-hoc networks. The dominators then take the role of server in the network. PBS uses four different roles for network nodes:

- INT-CANDIDATE – Initial status for a node running a specific application
- EXT-CANDIDATE - Initial status for a node not running this specific application
- DOMINATEE - Node is a direct neighbour of a zone server
- DOMINATOR - Node is a zone server

Initially, all nodes start either as an int-candidate or an ext-candidate. An int-candidate can eventually become either dominator and assume the role of zone server for the application or it can become dominatee. Because PBS calculates a dominating set, a dominatee always has a zone server as neighbour to which it can then connect. An ext-candidate is different because it does not run the client application. Hence, an ex-candidate node can never become a dominatee. It however can become dominator because of its network position and device capabilities. Ext-candidate nodes that are chosen as zone servers can receive the necessary server software through the SIRAMON framework.

PBS is round-based and operates in a distributed manner. In each round, every node executes three steps. Firstly, it broadcasts its current neighbour list to all neighbouring nodes. Secondly, it receives the neighbour lists from its neighbours and finally based on the received information it determines if its status has changed. The neighbour list contains a *fullconnected* status flag that dominator nodes can set if they have a link

to all nodes in the network. Nodes that have a dominator as neighbour automatically switch to dominatee role. Otherwise, the dominator status is assigned if one of the following conditions are true: 1.) if the node has the highest priority among all known nodes, 2.) if it has a neighbouring node with no connection to other nodes (leaf node), 3.) if fullconnected status has been reported by another dominator. The priority metric that PBS uses is basically identical to the metric we use for the server selection algorithm. If a node determines its role either as dominator or dominatee, PBS stops and remains dormant until a network link change is detected. In this case, a new round is started, so that PBS can adapt to network changes. Changes in the network, e.g. due to node mobility, can lead to situations where two dominator nodes end up as neighbours. In this case, the dominator with lower priority reverts back to dominatee role.

PBS employs a distributed mobility prediction method that is based on received signal-to-noise ratio to determine stable, long-term links. Clients calculate if a link is expected to be stable in the future and connect only to those servers to which they have a dependable link. From the description of PBS in [155], it remains unclear what happens if a client is faced with only unstable links to servers and what if any additional delay this restriction produces for clients that loose its server due to mobility. Generally, the priority based selection algorithm aims at finding a good approximation for a minimum dominating set of nodes in the mobile ad-hoc network.

Although PBS and our approach solve the same problem and have many similarities, they are some major differences in the design and the operation. First and foremost, is the integration of PBS into the SIRAMON framework. Given the current state of the mobile market with different manufacturers that create a number of heterogeneous devices running diverse operating systems, we do not expect all these devices to run a single common middleware like SIRAMON. Hence, we think that additional MANET nodes that forward data but do not assist in server selection have to be taken into account as well. Moreover, PBS considers all network nodes to be prospective zone servers as long as they fulfil some basic requirements like having sufficient capabilities and have links to multiple nodes. This means that the priority based selection algorithm does not require a node to run a particular application itself in order to be chosen as zone server for it. Though such a selection can be desirable from a networking point of view, it may not necessarily reflect the goals of the user as it consumes battery power and communication bandwidth for his device. We see no incentive for the user to relinquish his own resources to offer a service to others. Also, such devices may be switched off by the user because he is unaware of the special role of his device for the network. Our server selection algorithm selects only nodes as servers that participate in that specific application. Here, a user serves its own interest and is assumed to willingly give up some of its resources for other members of his group.

Secondly, PBS calculates a distributed set in which every node is a server or a direct neighbour of a server. In fact, it aims at achieving a good minimum distributed set approximation which in theory can offer suitable server selection if most nodes run a specific application or if they are distributed evenly in the network. If application nodes are focused on one part of the network only, a distributed set of servers is not ideal. In conclusion, PBS uses a network-driven approach and nodes chose their role based on neighbour lists and priorities. This is in contrast to our server selection algorithm which employs an application- and client-driven approach. Here, clients chose the best server in their vicinity themselves or in cooperation with their neighbours. Thus, zone servers are created on an as-needed basis anywhere in the network. In contrast to PBS which requires client and servers to be direct neighbours, our server selection algorithm also allows a distance of up to two hops between clients and servers when making their initial determination.

Another point is that our server selection algorithm supports different applications in the network running at the same time. Applications can be differentiated through an unique application ID in the server selection protocol. Thereby, multiple server selection instances run separately without any direct interaction. Thus, servers can be selected in a way that is specifically tailored for each individual application taking the capabilities and the distribution of mobile devices into account. A network-centric approach like PBS that looks for a dominating set would chose the same servers for multiple applications which can lead to overload situations and degraded services for all applications.

Finally during server selection, nodes that use PBS broadcast a full list of their neighbours together with additional fields like the neighbour's state and priority as well as the fullconnect status flag. As more nodes join the neighbourhood, the total number of neighbour list entries increases exponential because every node adds the new neighbour to its list. Our server selection algorithm broadcasts a small and fixed amount of data that is independently of the number of neighbours.

PBS can make use of the XCoPred prediction method to predict link stability in mobile ad-hoc networks. By preferring stable links, an application is more likely to be able to utilize its server which can lead to better application performance and user perception. While our server selection algorithm makes no distinction of link quality, methods like XCoPred can easily be integrated. Besides XCoPred, another metric for link stability may be a simple SNR threshold filtering mechanism that can be easily computed. This approach is used by some implementations of the AODV routing protocol, e.g. AODV-UU, to ignore links that are very weak and therefore possibly unstable.

4.5. Chapter Summary

In this chapter we presented the server selection algorithm as a novel approach for creating and choosing servers in mobile ad-hoc networks. It operates in three phases discovery, selection, and maintenance. Its fully distributed design allows for good scalability. Each node regularly broadcasts information about its performance and network connectivity to its neighbours in small announcement messages. With this information every node then decides for itself the best server for its application. The selected zone servers provide services for nodes in their vicinity thus ensuring a satisfactory network performance. Depending on application-specific requirements, global synchronisation between zone servers can be decoupled from any latency-critical path which allows good response rates for clients. The application is also able to create or remove zone servers as it sees fit as the server selection algorithm will automatically adapt to the new situation. As every node constantly monitors announcement broadcasts from their neighbours, information about new nodes is updated in the neighbourhood table. If the application requests the selection of a better server due to node movement, the server selection algorithm can rely on local information and is able to select a zone server with little delay.

We have implemented the server selection algorithm in the network simulator NS-2. By relying only on NS-2's functions to receive and send network traffic, our implementation is easily portable. A comparison with related work shows that a client-based approach is more suited to select well positioned servers in a mobile ad-hoc network. Furthermore, our server selection algorithm allows multiple applications to run their individual instances of our algorithm without any interference from each other, creating a server network that is individually tailored to every application's needs.

5. Quality of Service

The Internet today is a best-effort network that does not provide any guarantee regarding the quality of a service. Still, latency-sensitive applications are used in local as well as in wide area networks. Instead of using a complex QoS scheme, wired IP networks are over-provisioned by design to allow decent quality for all users and applications. This approach, however, cannot be applied in the same way to wireless radio networks. First of all, current radio technology can only offer a small portion of the network bandwidth that is available for wired networks. Secondly, a wireless transmission occupies a defined frequency spectrum and a certain area around the transmitter for the duration of the transmission. Extending wireless network bandwidth is therefore not easy given the already crowded radio frequencies. This is in contrast to wired networks where signal propagation is confined to a single cable and bandwidth can be easily extended by deploying additional lines. While over-provisioning can also help in wireless networks, additional strategies need to be employed when latency-sensitive applications from the Internet cross over to the wireless world. Moreover, using such applications in mobile ad-hoc networks is additionally challenging because of outside interference and node movement.

In this chapter we discuss the feasibility and the limits of using latency-sensitive application over multi-hop wireless ad-hoc networks. We start by making single hop measurements using existing off-the-shelf hardware to analyse real-world wireless network traffic. Specifically we examine round-trip time latency, latency variation and packet loss for different distances and situations. By simulation we then determine the most suitable ad-hoc routing protocol for applications. We follow up by looking at various QoS improvements for medium access, routing as well as queuing which we have implemented in the network simulator NS-2. Through simulation of a multi-hop scenario, we analyse the effect of those improvements. Finally, a summary concludes this chapter.

5.1. Preliminary Evaluation

In Chapter 3 we discussed capacity, fairness and latency of wireless LANs from a theoretical point of view and introduced MANET routing protocols. We now follow up this consideration with measurements and examine latency, latency variation and packet loss in a real WLAN network. We thereby specifically look into the individual factors that have a significant contribution to latency. Also we take a look at differences to wired networks that can effect algorithms of existing latency-sensitive applications. Furthermore we present network simulations to find a MANET routing protocol that is able to provide low latency multi-hop data traffic. Finally, we summarize our results and draw conclusions and recommendations for latency-sensitive applications.

Although an end-to-end measurement in a multi-hop scenario would have been better suited to analyse network parameters for latency-sensitive applications in MANETs, our following experiment focuses on a single hop only. The reason for this is that our notebooks were not time synchronized during our experiments and thus the correlation of events was a complex task even for two devices. Therefore, we decided to look at a real single wireless link in detail and then move on to more complex multi-hop experiments in a simulator later on where we have no problems synchronising events.

5.1.1. Single Hop Measurements

For our measurements we used two 1,8 GHz Intel Pentium M notebooks in an outdoor urban scenario with distances between both devices of 1 and 5 meters and then continuing up to 50 meters. All notebooks were equipped with CardBus WLAN cards that uses an Atheros WLAN chip, an internal antenna and supports IEEE 802.11 a/b/g operations. The maximum achievable communication distance in our scenario was 193 m. To emulate network traffic of a latency-sensitive application we used ICMP Echo request and response packets with different packet sizes of 104, 576 and 1400 bytes and a send rate of 10 pps. Each measurement lasted ten minutes with 15 seconds adjustment time at the beginning to allow the rate adaptation mechanism to adapt to the new environment. Furthermore, we repeated our measurements and used a third notebook to insert additional background traffic to the scenario. This background traffic comprised a single TCP stream that continuously transmitted 1500 bytes frames during the measurement.

We used a modified version of the multi-band Atheros driver for WiFi (MadWifi) WLAN module for Linux[156]. MadWifi uses the SampleRate algorithm for rate adaptation and multi-rate retries that allows control of how the transmission rate is

adjusted in case of retransmissions. By default MadWifi resends frames up to ten times. In order to be able to record events and timestamps at a fine grained level, we modified the MadWifi driver to log events such as frame queuing, transmissions and receptions of data frames as well as link layer acknowledgments . Because of the strict timing requirements of the WLAN protocol, some of these functions are done independently of the operating system by the Atheros hardware itself. We therefore added an interrupt-based callback routine where the hardware notifies the driver that a frame has finally been sent or received. In an initial evaluation, we found that the overhead caused even for a significant number of interrupts was negligible for our measurement. Furthermore, the Linux kernel's standard time base does not offer high-resolution timing information which we need for time stamping WLAN protocol events. We therefore implemented our own method which is based on the CPU's time stamp counter (TSC) to give us high precision time information. Because the TSC is increased with the CPU internal clock rate, it allows nano-second granularity. Naturally, because of unpredictable hardware-level events, our implementation cannot guarantee such precision. However, through experimentation we found that our solution can offer a reliable time measurement in the area of $50 - 100\,\mu$s which is sufficient for our experiment. We also gathered additional information from the driver and from the Atheros hardware for each event. This information includes the requested, sent and received data rates, the signal strength and the number of retransmissions.

Table 5.1 shows the results of our experiment at three selected distances. At each distance we show measurement results for different packet sizes with and without additional background traffic. For all scenarios, the round-trip time and its variation remains well below 150 ms which is our threshold for latency-sensitive applications. For the variation of round trip time, we can see values between half and one time the value of the round trip time itself which is quite high. In related work, Armitage and Stewart[157] analysed the relation between delay and delay variation in the Internet and found a high correlation between both values with the delay variation being generally one fifth or less of a path's latency. Although, the results from Table 5.1 indicate generally low RTT variation for single hop transmissions, adapted delay variation compensation methods may be required to compensate for larger delay variations in wireless networks.

Finally, the results look a little bit different for packet loss. Firstly, we notice an anomaly of high packet loss at 50 m distance with background traffic. In our scenario, the first node that sends our data and the node that generates the background traffic are close together. Thus, communication between them usually is done at the fastest transmission speed of 54 Mbit/s. However, at 50 m distance the second node is unable to detect high-speed transmissions between the two remote nodes which creates

Packet size						
Dist.	avg. RTT		avg. RTT variation		Packet loss	
	w/o bgtfc	w/ bgtfc	w/o bgtfc	w/ bgtfc	w/o bgtfc	w/ bgtfc
104 byte packets						
1 m	0.74 ms	6.86 ms	0.74 ms	5.62 ms	0.21%	0.15%
25 m	0.62 ms	9.07 ms	0.46 ms	7.04 ms	0.98%	0.72%
50 m	1.40 ms	7.07 ms	0.73 ms	3.84 ms	1.76%	22.95%
576 byte packets						
1 m	0.84 ms	6.94 ms	0.66 ms	5.57 ms	0.36%	0.21%
25 m	1.04 ms	11.58 ms	0.70 ms	7.22 ms	3.84%	2.02%
50 m	2.85 ms	13.88 ms	1.38 ms	6.48 ms	1.37%	25.37%
1400 byte packets						
1 m	0.97 ms	7.09 ms	0.41 ms	5.42 ms	1.04%	0.05%
25 m	3.48 ms	20.38 ms	1.67 ms	12.24 ms	0.62%	2.21%
50 m	5.27 ms	26.65 ms	2.52 ms	12.92 ms	2.98%	17.41%

Table 5.1.: Measurement results as seen by the application

a transmission-rate dependent hidden terminal problem. This problem is usually dealt with by using the RTS/CTS mechanism of IEEE 802.11 which we discuss later in the simulation section of our QoS improvements. For other distances and for scenarios without background traffic, the packet loss remains well below 5%, our threshold for latency-sensitive applications. Nontheless, if we extrapolate our data for multi-hop communication, a problem becomes apparent as packet loss is a multiplicative factor. Table 5.2 shows the maximum average acceptable packet loss per link that keeps the end-to-end packet loss below 5 %. The values are calculated by using the following equation with p_{loss} being the link loss on all individual links:

$$\begin{aligned} 1 - (1 - p_{\text{loss}})^{\text{hops}} &= 0.05 \\ \widehat{=} \qquad p_{\text{loss}} &= 1 - \sqrt[\text{hops}]{0.95} \end{aligned}$$

If we compare these numbers with our measurement results, our data suggests that for small packets of 104 bytes the maximum achievable distance in a multi-hop wireless network is between three and five hops if the end-to-end loss is to be kept below 5%. The distribution of packet loss in our experiments confirmed prior results by Carvalho, Angeja, and Navarro[77].

Figures 5.1(a) and 5.1(b) show the measurement results for 25 m distance as a cumulative density function of round trip time. Because of the event logging of our modified

Hops	1	2	3	4	5
Maximum loss per link	5.00%	2.53%	1.70%	1.27%	1.02%

Table 5.2.: Calculated maximum link loss for an end-to-end packet loss rate of 5 %

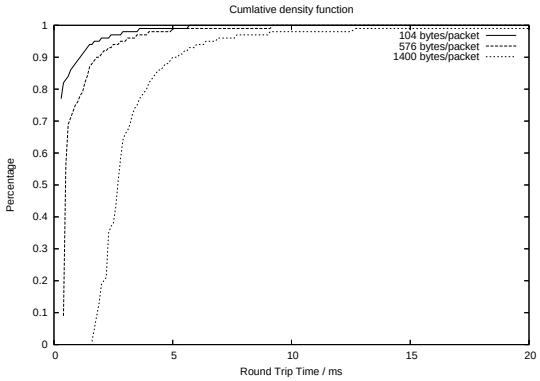
driver we were to determine the one way delay for each data frame. By using our modified WLAN driver and a logfile of all queuing, sending and receiving events of data packets as well as acknowledgements from our measurements, we can determine the individual end-to-end delay of a data frame. Figure 5.1(c) presents a comparison of one-way delays of corresponding request and response packets. It shows that for larger distances, the common concept of one-way delay being half the round trip time is not valid anymore. Again this behaviour is exhibited because WLAN allows different transmission speeds that are determined individually by the sender. In our experiments, we found that the majority of request and response packets were sent at different transmission speeds. Although, these differences usually mean one node is using the next faster transmission speed than the other, it has a measurable effect on one way delay if the transmission is sent over greater distances. Here, the total difference in transmission speed can be significant (1 MBit/s vs. 2 Mbit/s) whereas at shorter distances it is almost negligible (48 MBit/s vs. 54 MBit/s). For multi-player games that rely on being able to calculate the one way delay of a player to order game events, this effect can have an impact on game fairness. Figure 5.2 shows the transmission rates for request/response pairs. An 'X' in the figure denotes that at least one pair used this combination of transmission speeds and the darker the background, the more often this combination occured in our measurements.

We also look at packet retransmissions and to what degree they contribute to successful packet delivery which is shown in Figure 5.1(d). Although the usefulness of packet retransmissions varied slightly for different packet sizes and distances, the sweet spot is generally at six retries after which the probability of successful packet delivery drops below 50%. Hence, we recommend reducing the number of retransmissions to six or less for latency-sensitive applications in order to reduce unnecessary load on the wireless network.

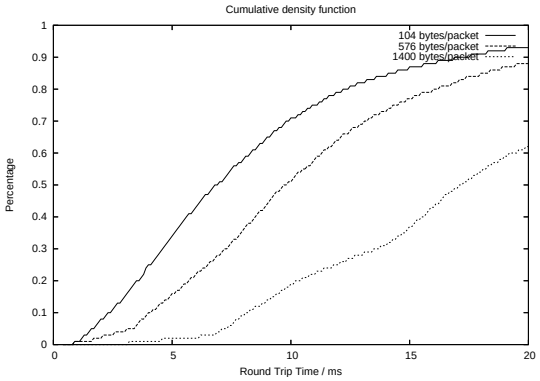
A thorough composition analysis of the round trip time as seen by the application for measurements without background traffic and a packet size of 104 bytes is shown in Table 5.3. We have decided only to look at the results for the smallest packet size as they are the most relevant for latency-sensitive applications. And because we focus on round trip time, only those request/response pairs have been taken into account that were transmitted successfully possibly with some link-layer retries. Nevertheless,

	Name	Distance 1m			Distance 25m			Distance 50m		
		Delay	Std.-Dev.	Contrib.	Delay	Std.-Dev.	Contrib.	Delay	Std.-Dev.	Contrib.
Initial pkt	Tx Request	101.8μs	0.3μs	13.8%	113.5μs	1.4μs	18.4%	270.7μs	2.0μs	19.3%
	Tx Ack	40.0μs	0.0μs	5.4%	40.2μs	0.5μs	6.5%	161.8μs	0.4μs	11.5%
	Tx Response	102.6μs	0.6μs	13.9%	123.3μs	1.6μs	20.0%	305.3μs	4.3μs	21.8%
	Subtotal RTT Init	244.4μs		33.2%	277.0μs		44.9%	737.8μs		52.6%
	Avg. Tx tries per packet	request 1.008, response 1.008			request 1.029, response 1.048			request 1.057, response 1.239		
Retries	Tx Request Retry	0.8μs	11.1μs	0.1%	3.5μs	20.7μs	0.6%	15.1μs	20.5μs	1.1%
	Tx Response Retry	0.7μs	10.8μs	0.1%	6.8μs	27.4μs	1.1%	73.4μs	30.0μs	5.2%
	Subtotal RTT Retry	1.5μs		0.2%	10.3μs		1.7%	88.5μs		6.3%
Overhead	Channel & Contention	456.7μs	87.8μs	62,0%	295.5μs	54.6μs	47.9%	541.6μs	58.6μs	38.6%
	Kernel	32.6μs	1.1μs	4.4%	32.2μs	0.9μs	5.2%	31.7μs	1.3μs	2.3%
	Subtotal Overhead	489.3μs		66.4%	327.7μs		53.1%	573.3μs		40.9%
	Total RTT	735.2μs		99.8%	615.0μs		99.7%	1,399.6μs		99.9%
	Measured RTT	736.8μs	69.1μs	100.0%	616.6μs	38.4μs	100.0%	1,401.4μs	38.2μs	100.0%
	Error	1.6μs		0.2%	1.6μs		0.3%	1.8μs		0.1%

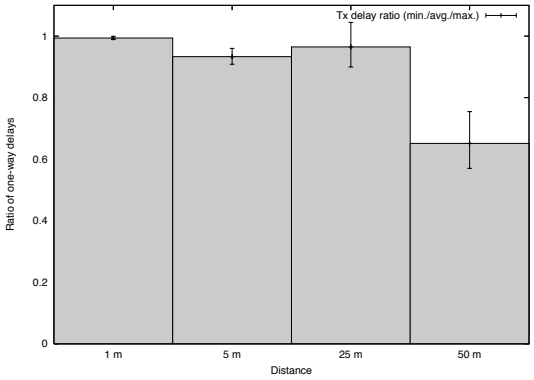
Table 5.3.: The sources of delay in wireless LANs – Contributing factors to the average round trip time for the scenarios without background traffic



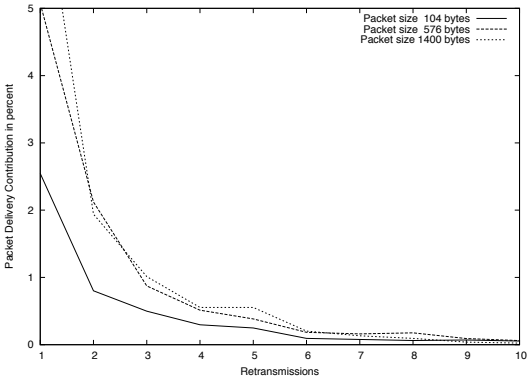
(a) Latency CDF with no background traffic



(b) Latency CDF with background traffic



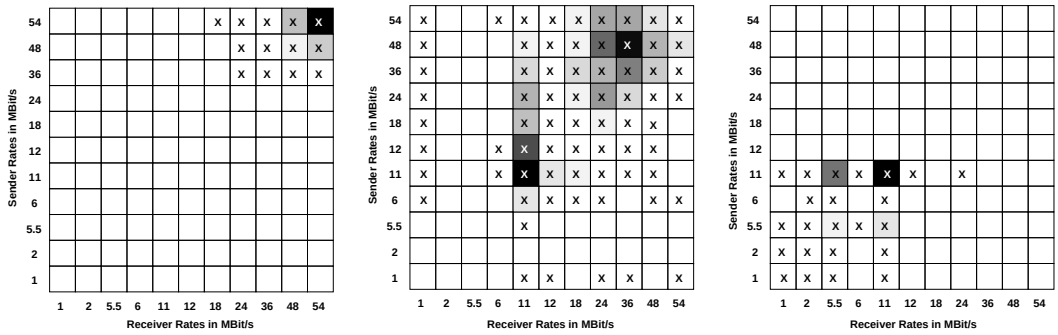
(c) One way delay comparison (various distances)



(d) Packet Delivery Contrib. of Retransmissions

Figure 5.1.: Measurement results at 25 m distance

packet loss in these scenearios were generally below two percent as can been seen in Table 5.1. The numbers shown in Table 5.3 are the results of an in-depth analysis of the event logs from our measurement. Unfortunately, it was impossible to measure transmission and waiting times directly because these functions are done within the WLAN hardware and are not accessible by software. Instead we used information about transmission and reception events as seen by our MadWifi driver on both nodes to determine the time spent for sending data frames and low-layer acknowledgments as well as queuing delays that come from channel access and WLAN protocol overhead. We then used information about transmission speed, frame coding and retries to calculate the transmission delay for each individual frame. Finally, we added up all our results for each individual request/response pair and compared our results with the round trip time as seen by the application. We omitted propagation delay completely as it is not relevant at the distances in our scenarios and our results show only a small error. A more detailed description of how we measured and calculated these values as



(a) Distance 1m (b) Distance 25m (c) Distance 50m

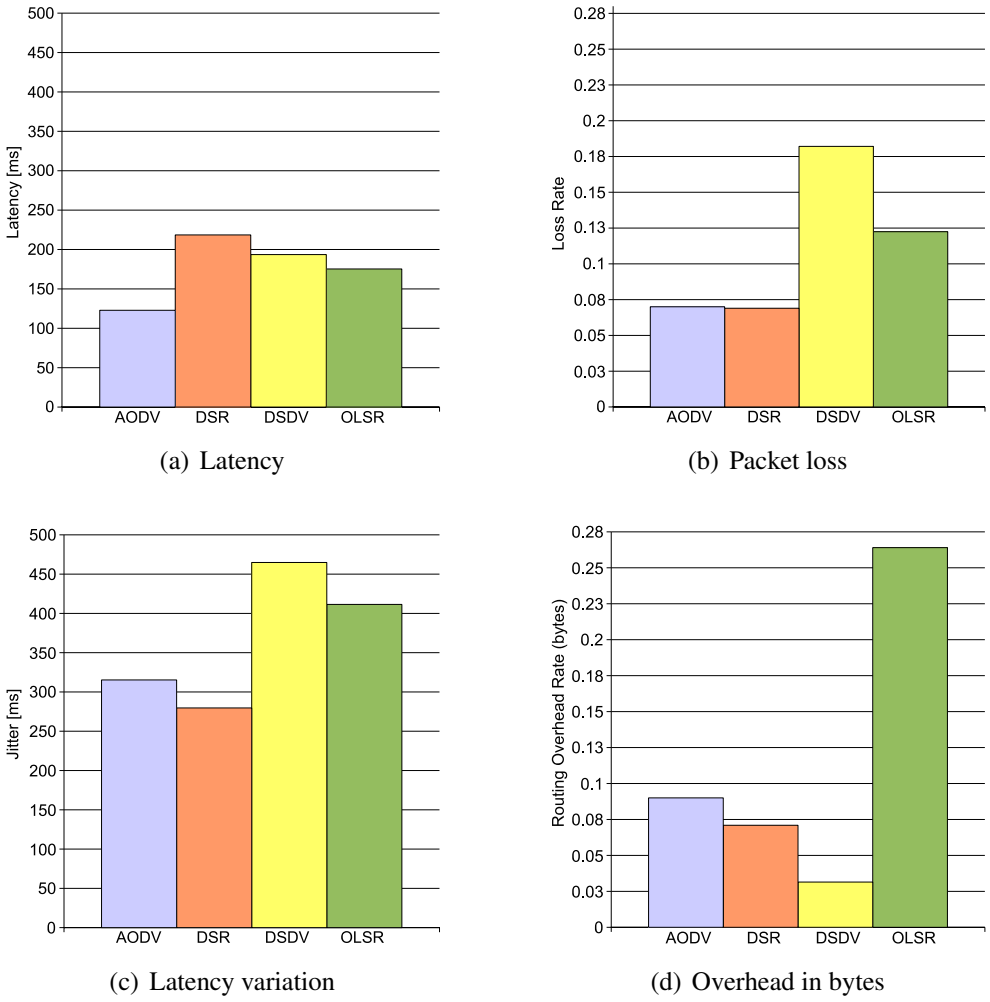
Figure 5.2.: Sender/Receiver transmission rates for corresponding ICMP echo request/response packets

well as an example can be found in Appendix A.2.

Our break-down shows that the transmission delay is only responsible for between one third and half of the round trip time. The remaining time is mainly taken by waiting for a free wireless channel and the contention period. As expected in a scenario with no significant background traffic, transmission retries have no noticable effect on the average round trip time. This is different from our results for larger packets where retransmissions can take up to 20 % of the round trip time and the channel and contention overhead is only about 15 %.

5.1.2. Simulations of Routing Protocols

Routing protocols offer an important service in mobile ad-hoc networks as they discover and maintain routes between mobile devices. They can also influence the application performance due to their route setup delays and protocol overhead. We have done a preliminary evaluation of four different MANET routing protocols in NS-2 to find a good candidate that is able to support latency-sensitive applications. The cost of a routing protocol can be divided into two major parts. The network cost comprises communication overhead for sending and receiving routing packets and the node cost contains processing requirements to calculate a route through the network as well as storage requirements to save routing table and protocol-specific data. Obviously, both factors are not fully independent from each other. For the purpose of our preliminary evaluation, we looked at the network cost of each routing protocol only. We selected the Ad-hoc On-Demand Distance-Vector protocol (AODV[93]), Dynamic Source Routing (DSR[95]), Destination-Sequenced Distance Vector (DSDV[90]) and Optimized Link State Routing (OLSR[91, 92]) for our evaluation which are all well-



(a) Latency

(b) Packet loss

(c) Latency variation

(d) Overhead in bytes

Figure 5.3.: Routing protocol comparison results (from [160]

known routing protocols for MANETs. For DSR and DSDV, we used the existing implementation in NS-2. For AODV and OLSR, we used the implementation from Uppsala University (AODV-UU[158]) and an implementation from Murcia University (UM-OLSR[159]) respectively.

Our evaluation uses a scenario with 20 nodes that move at walking speed between zero and three meters per second according to the random waypoint mobility model. We randomly select 15 nodes that create unidirectional CBR traffic with 20 pps and a packet size of 64 bytes. Figure 5.3 shows the average results of 60 different simulations that each lasted 10 minutes.

To achieve a maximum round-trip time of 150 ms for latency-sensitive applications, one-way latency must on average be kept below 75 ms if we assume a symmetric dis-

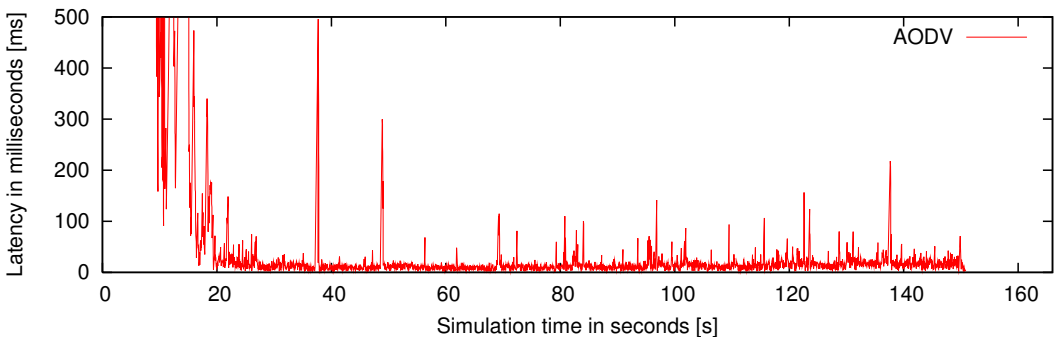


Figure 5.4.: Typical AODV delay distribution (from [160])

tribution of delay. While such an assumption is not generally valid in mobile ad-hoc networks, it is an appropriate premise for our simulations as we have select sending and receiving nodes randomly and only look at average values for latency. With an average one-way latency of 125 ms, AODV shows the best result in our simulations as can be seen in Figure 5.3(a). Nevertheless, it stays above the required value for latency-sensitive applications and the other three protocols achieve even much higher delays. Regarding packet loss, no protocol is able to achieve less than 5 % loss but AODV and DSR come in close. Also noteworthy is the fact that latency variation is high for all protocols reaching values between three and four times one-way latency with both proactive protocols being at the top of the list. Finally, OLSR shows a very large overhead even when compared with DSDV which also is a proactive protocol. The reason for this is that the size of OLSR messages grow linearly with the number of nodes. More on the overhead of OLSR and how to deal with it can be found in [161] and [162]. From the results of our preliminary evaluation we have chosen AODV as the best candidate for latency-sensitive applications because it achieved the lowest delay and is on par with DSR on packet loss. In the following section we discuss methods to improve AODV's performance as well as network performance for latency-sensitive applications in general.

Reactive protocols only keep routes to other nodes which are being used. Thus, sending traffic to a new destination requires a route discovery process which yields in an initial route setup delay for reactive protocols. Depending on network load and the size of the network, this initial delay can be quite large. Figure 5.4 shows an example of this setup delay from an exemplary UDP data stream of 150 seconds duration from our simulation. Here, we notice that the initial delay is high when compared to the average delay for the whole duration. The reason for this is the mentioned route setup as well as the fact that the network load is high during this period as most nodes do their route

discovery. For the remaining time, one-way latency is much lower generally around 20 ms. We can also observe three larger values above 200 ms and many lower peaks generally below 150 ms which are the result of re-routing efforts and retransmissions. For latency-sensitive applications, the initial setup delay is not as important as delay during data transmission. Still, it has an impact on user perception as the actual setup may take several seconds.

5.1.3. Conclusion

In this section we presented a preliminary analysis of delay in WLAN and determined a suitable routing protocol for mobile ad-hoc networks. Through experimentation, we analysed the source of latency for single hop WLAN communication in a specific scenario. We then continued our preliminary analysis by simulating a multi-hop wireless networks in which we compared the performance of various MANET routing protocols. From our results we have shown that WLAN is generally able to support latency-sensitive applications. We discussed that a major part of the total end-to-end delay is spent on channel access and contention. Thus, we can safely conclude that improved medium access control strategies like e.g. 802.11e can reduce this delay for high priority traffic. Delays from network link changes and subsequent re-routing efforts in a MANET as well as WLAN retransmissions can result in higher latency up to a point which exceeds the maximum latency for interactive applications. Furthermore, our measurement and the simulations have shown that applications should expect increased latency variation in radio networks up to several times the value found in wired networks. Regarding packet loss, WLAN achieves packet delivery ratios which are suitable for latency-sensitive applications. However in MANETS, our preliminary evaluation suggests that only a couple of hops can be covered within the maximum packet loss constraints. The reason for this is the multiplicative effect of packet loss over multiple hops. We have also shown that a transmission rate dependant hidden terminal problem exists which can impair latency-sensitive communication significantly especially when the wireless network is used close to its maximum capacity. Our measurement has shown that the break-even point for the maximum number of retransmissions is six. For latency-sensitive applications that prefer packet loss over high-latency data, the number of retransmissions should be reduced further. Finally, we have chosen AODV as our routing protocol because of its low overhead and the fact that its performance comes closest to the requirements for interactive latency-sensitive applications. In the following section, we discuss several methods to improve routing and network performance for latency-sensitive applications.

5.2. QoS Improvements

Previously we have shown that even while AODV shows the best performance among our set of MANET routing protocols, it fails to achieve the necessary performance for interactive latency-sensitive applications in multi-hop networks. In this section we aim at increasing the network performance by introducing new as well as existing QoS-related enhancements to AODV and MANETs. We evaluate all approaches with NS-2 to assess their individual as well as combined impact on data traffic and general network performance. Furthermore, we look into the effect that those methods have on non-interactive background data traffic. But first we take a look at related work where many QoS enhancements for mobile ad-hoc networks or wireless networks in general have been discussed.

5.2.1. Related Work

Quality of service support for mobile ad-hoc networks can be achieved on different layers. Medium access control protocols can achieve service guarantees by controlling when a mobile node can access the wireless channel. Other approaches focus on the networking layer where QoS-aware routing protocols discover only routes that are able to support the network requirements of individual applications. Finally, QoS frameworks define different QoS classes and mappings, resource reservation and signaling as well as admission and rate control mechanisms. Also, they use QoS-aware queuing strategies to support prioritisation and differentiate various flows. They often operate on top of existing non-QoS MAC and/or routing protocols. We give a brief overview of selected approaches from each of these three categories.

QoS Medium Access Control Protocols

Many mobile ad-hoc networks use wireless LAN as radio and medium access technology. The standard WLAN protocol family is contention-based and is therefore not able to offer any service guarantees. In 2005, two additional coordination functions were published as IEEE 802.11e[44] that offer traffic prioritisation which can be helpful in situations where a single service e.g. Voice-over-IP needs to be preferred over other traffic. We already discussed this WLAN extension briefly in Chapter 3. Based on IEEE 802.11e, You, Yeh and Hassanein proposed DRCE[163], a high throughput QoS MAC protocol for dual-channel wireless ad hoc networks. The first channel is used to communicate request/clear to send (RTS/CTS) frames as well as small data frames. Larger frames are transmitted on the second channel after successful signaling

on the first. DRCE introduces an additional Ensure to Send (ETS) frame to allow the transmission of sender and receiver schedules. Additionally, DRCE proposes negative acknowledgments to allow for a timely rescheduling by the sender. The authors show in simulation that they can achieve an average delay that is twenty to forty percent lower than with IEEE 802.11e.

Lin and Gerla[164] suggested MACA/PR an extension of the common multiple access with collision avoidance scheme with piggy-backed reservations. Such reservations are transmitted together with an initial RTS or previous data frame and are stored in a reservation table by neighbouring nodes. MACA/PR differentiates between two classes, non real-time and realtime traffic to which it can give bandwidth guarantees. The authors present MACA/PR as an alternative to TDMA-based approaches for multi-hop networks which keeps the simplicity of asynchronous data transmission.

Becker, Gotzhein, and Kuhn proposed MacZ[165], a QoS MAC layer specially designed for mobile ad-hoc networks. MacZ uses a multi-hop tick synchronisation scheme that allows network wide allocation of transmission slots. It employs so called macro slots that consist of phases that are contention-free followed by a contention-based phase that takes priorities into account. The slot boundaries are globally synchronized. Hence, the tick synchronisation scheme does not work fully distributed. Traditional medium access protocols operate on the data link layer. As such, they take only the transmission from one node to the next into account and need additional support from higher layers to guarantee service quality over multiple hops.

QoS Routing

On the network layer, several QoS protocols have been proposed. We show a selection of some well-known approaches which are relevant for our application. Sinha et.al[98] introduced CEDAR, the core-extraction distributed ad-hoc routing algorithm. It calculates an approximated dominating set which is then used as backbone. Multi-hop routing over longer distances is done via this backbone and its core nodes support quality of service. This approach reliefs leaf nodes of the complexity of handling QoS traffic themselves.

In 2003, Perkins and Royer introduced QoS extensions for the commonly used ad-hoc distance vector routing protocol (AODV). It allows the addition of QoS requirements to route request packets and intermediate routes would only answer such requests if their route could fulfill the necessary prerequisites. Furthermore, all nodes need to be aware of their average latency and jitter values and would substract these values from the corresponding QoS requirement before forwarding a route request to neighbouring nodes.

Xue and Ganz[166] proposed the ad hoc QoS on-demand routing protocol (AQOR) that operates in a similar manner. However, AQOR exchanges periodic messages between neighbouring nodes to determine the available channel bandwidth. Thus, it allows to take temporary resource reservations in the same collision domain into account.

Another addition to AODV was proposed by Sakurai and Katto[167] that adds an multi-path extension to AODV. If multiple routes to the same destination are maintained, additional delays can be avoided if a route fails. The authors also use additional metrics such as delay and disjointness for route discovery.

Yang and Huang[168] suggest the dynamic multi-path source routing protocol (DMSR) which tries to find a ideal number of routing entries to a destination. These routes are then used simultaneously to balance the network load. The authors show that this approach can increase the packet delivery ratio in mobile ad-hoc networks.

QoS Frameworks

Many QoS frameworks for mobile ad-hoc networks are based on existing work from wired networks. Xiao et.al[169] introduced a flexible QoS model for MANETs (FQ–MM) that combines features from the well-known IntServ[170] and DiffServ[171, 172] models. FQMM provides service differentiation per flow for high priority traffic and per class for traffic with lower requirements. As a hybrid approach it shares similar benefits and problems of the original QoS models that it is based on.

Zhang et.al[173] proposed INSIGNIA as a light-weight stateful QoS framework. It employs an in-band signaling protocol which uses an IP option for soft-state resource reservation at intermediate nodes. INSIGNIA uses per flow timers to implicitly release reservations if no data packets have been received for some time.

He and Abdel-Wahab[174] suggested HQMM, a hybrid QoS model for mobile ad-hoc networks. It adds a per-class model to INSIGNIA, effectively creating a hybrid between INSIGNIA and DiffServ.

Ahn et.al[175] proposed service differentiation in stateless wireless ad-hoc networks (SWAN). It distinguishes two classes, low priority data which is rate controlled and high priority traffic. SWAN uses probe messages with which the source probes the available bandwidth to the receiver as a means of admission control for high priority traffic.

Finally, Chen, Shah, and Nahrstedt[176] introduced the integrated mobile ad-hoc QoS framework. Their approach uses location-aware routing to predict node movement and

network partitioning. A central point of this architecture is a middleware which shares information across layers about the status of other nodes and the network in general. It also provides QoS signaling between layers, performs bandwidth estimation, management and rate adaptation.

In this thesis we look at latency-sensitive interactive applications to work with commodity WLAN hardware and drivers. Therefore, we do not to take any MAC protocol enhancement into account that changes the behaviour of WLAN's DFWMAC. Regardless, medium access methods that are specifically designed for mobile ad-hoc networks and are able to differentiate traffic classes can be of benefit for our applications. But for the purpose of this thesis, we regard QoS MACs as future work. Another important aspect is that mobile devices have power, memory and energy constraints that require low complexity approaches that operate in a distributed and scalable fashion. In the following section we will therefore take a closer look at individual ideas from related work.

5.2.2. Methods

In this section we present nine methods to improve the performance of latency-sensitive applications. These approaches can be categorized in three different areas and are shown in Table 5.4. First of all enhancements for medium access control which reduce overhead to determine local reachability information. Secondly, routing improvements which contain approaches to rapidly repair routes and to determine the quality of network links. And finally queuing strategies which offer prioritisation and shaping of traffic. We focus on methods which can be implemented with current hardware and therefore only take a look at methods which require no modification of WLAN hard- or firmware and can be applied by changing the software of the operating system or the WLAN driver. Thus, QoS enhancements such as 802.11e are not considered. On the following pages we present a brief overview of all nine QoS enhancements which we have implemented in NS-2. We then discuss their performance in a simulation with latency-sensitive and background traffic.

RTS/CTS Adaptation

Request-to-send (RTS) and clear-to-send (CTS) control packets are standard features of the DFWMAC. They are send before a data packet to reserve the wireless channel at the receiver and therefore can reduce the hidden terminal problem. On the downside, sending RTS/CTS packets increases the general network overhead. Applications that send many small packets, such as transmissions of latency-sensitive applications,

Category	Feature
MAC enhancements	RTS/CTS Adaptation Neighbour Detection Broken Link Detection
Routing enhancements	Signal Strength Monitoring Local Repair Backup Route
Traffic Management & Prioritisation	Priority Queuing Early Discard Neighbour Aware Rate Control

Table 5.4.: QoS enhancement methods

this overhead can have a noticeable impact on network performance. Thus, in order to be able to choose a trade-off between overhead and hidden terminal resilience, the RTS/CTS threshold defines the minimum data packets size for which the RTS/CTS mechanism is used. The standard NS-2 WLAN configuration uses a threshold of zero bytes which enables RTS/CTS for all packets. For RTS/CTS adaptation, we use a setting of 1,000 bytes so that traffic from latency-sensitive applications is sent without this additional overhead while larger packets still retain this method to cope with hidden terminals.

Neighbour Detection

In mobile ad-hoc networks, nodes maintain a list of neighbouring nodes for reachability and routing purposes. Keeping this list up-to-date within the distance of a single hop is cheap because it can be achieved with periodic broadcast messages by every node. Such periodic message are already used in wireless network for different purposes. On lower layers, WLAN nodes in ad-hoc mode send out periodic broadcasts to announce the ad-hoc network name and parameters which can be used to augment reachability information. Also routing protocols like AODV periodically send HELLO messages to detect which nodes are directly reachable. Finally, our server selection algorithm uses periodic broadcasts for neighbour discovery. Using not many but a single common neighbour detection scheme can help reduce broadcast traffic especially in densely populated mobile networks. Although, such messages are generally not very large, broadcast messages bear the penalty of being sent at low transmission speed and accordingly use up valuable resources. Unfortunately, NS-2's simple 802.11 MAC layer does not use such low layer beacons in ad-hoc mode, which prevented us from

including a common neighbour detection scheme in our simulation. Instead, we have implemented a beacon-based neighbour discovery for AODV and the Linux MadWiFi WLAN driver in [160].

Broken Link Detection

AODV regards a link to one of its neighbours as broken if it does not receive a HELLO packet during the time interval in which usually three such packets are send. As periodic broadcast intervals are usually a trade-off between responsiveness and overhead, it can take some time before the routing protocol discovers that one of its neighbours is not reachable anymore. With WLAN however, there is a another way to detect broken links as all unicast packets need to be acknowledged by the receiving node. This link-layer feedback is available considerably faster and without any additional overhead. It also has the advantage of tying link status information closer to the actual data instead of relying on separate broadcast transmissions which are sent in the other direction. In case that a packet gets lost during transmission, the WLAN MAC automatically retransmits it thus ensuring that negative information is only passed on to the routing protocol if a packet is dropped.

Signal Strength Monitoring

The signal-to-noise ratio (SNR) is an important characteristic of data communication to determine whether a signal can be recognized and decoded correctly by the receiving node. Successful reception of a data packet thereby depends on the SNR and the sensitivity of the receiver which can be seen as a SNR threshold for a particular device. In reality however, no clear prediction can be made for signals close to this threshold as discussed by Lundgren, Nordström, and Tschudin[177]. They state that a grey zone exists around this threshold in which the reception probability varies due to fluctuating links. Furthermore in grey zones, small packets like for example routing packets have a greater probability to be transmitted successfully which often leads to the creation of new routes that have a high packet loss for larger data packets. To avoid the creation of routes on unstable links, the signal strength of a received data transmission can be used to identify grey zones. In detail, an AODV implementation on any node would not forward an AODV route request packet that has a weak SNR value. By introducing an SNR threshold in AODV that lies well outside the grey zone, route quality can be improved. Unfortunately again, we had to exclude this feature from our preliminary evaluation because NS-2's propagation models do not export the SNR values for individual packets. However, a detailed evaluation of this feature can be found in [177].

Local Repair

Local repair is another AODV improvement that was proposed by RfC 3561[94]. With AODV, an intermediate node that notices a broken link along a route sends a route error message back to the sender. The sender then restarts a route discovery process to find a new route to its destination. With local repair, the intermediate node itself tries to find a new route to the destination to replace the broken part of the previous route. During route discovery, the intermediate node buffers packets for the destination. Local repair can have the advantage that a replacement route is found faster because errors do not need to be reported back and only a part of the route needs to be rediscovered. On the other hand, local repair may produce a longer than necessary route depending on the position of the intermediate node in the network. In any case, the sender can always probe for a new route if it thinks that routing needs to be optimized. For latency-sensitive interactive applications, local repair can help to reduce intermittent interrupts in connectivity which are harder to mask to the user than higher latency.

Backup Route

Another approach to deal with broken routes is to keep one or more backup routes to the destination in case the primary one fails. Once a link failure is detected, backup routes can speed up route recovery time because ideally no further route discovery is necessary. Also, because the backup route is usually kept at the sender, it can decide which routes are suitable as a backup route. One problem of this approach is finding a backup route that is independent from the primary one. Also, because backup routes are discovered and then stored for later use, they may not be valid anymore when they are needed. Finding backup routes with AODV comes at no additional cost because AODV broadcasts its route requests. Thus, all routes with the maximum given hop-count are discovered.

Priority Queueing

Priority queuing is a traditional QoS mechanism to differentiate traffic at intermediate nodes. We use a classifier to distinguish between low, medium and high priority traffic. Data from real-time or latency-sensitive applications are put in the high priority queue. Routing traffic in general is assigned the medium priority queue and all other traffic goes to the low priority queue. The design of our priority queue implementation for NS-2 can be seen in Figure 5.5. The top two queues are short and optimized for low-latency traffic whereas the low priority queue is designed for high throughput. For the

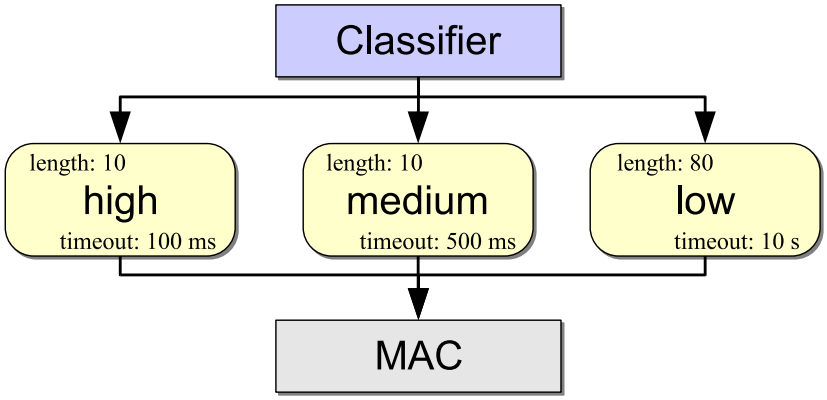


Figure 5.5.: Traffic differentiation with priority queues (from [160])

purpose of our analysis, we use the type of service field in the IP header to distinguish the type of data packets.

Early Discard

Interactive data traffic is sent with high packet rates to be able to react rapidly to changes in the application. Thereby succeeding packets often update data sent only a few packets earlier. This means that delayed packets are useless to the application because their data is obsolete. Still in situations where a link gets overloaded, interactive data is queued at intermediate nodes alongside other traffic to be transmitted over the congested link. With early discard, we put an upper boundary to the queuing time of interactive data packets because their late transmission is often not useful to the application anymore. As a side effect and as traffic is removed from the queue, newer packets have an increased chance of being transmitted in time. In our implementation we use different timeouts for our three priority queues which reflect their individual requirements (see Figure 5.5). Queuing time is calculated locally at each node at enqueuing and dequeuing events.

Neighbour-aware Rate Control

Although some newer WLAN hardware supports the 802.11e QoS extension, applications often cannot use these benefits for their latency-sensitive traffic because the WLAN driver, the network layer, or the operating system in general lacks QoS support. Thus, in high traffic situations, low-latency traffic can get delayed while other low-priority traffic is being transmitted. To overcome this problem for our evaluation,

we implemented a simple form of cooperative QoS support for distributed access networks at the routing layer. With neighbour-aware rate control, nodes voluntarily place an upper bound on low-priority traffic if they detect that a neighbouring node is sending high-priority traffic. We have employed an unused bit in the AODV message header to indicate nodes that send high-priority traffic. This bit is used in AODV's route request, route response as well as its periodic HELLO messages. For sake of simplicity, we did not use a full QoS classifier to distinguish packets into high- and low-priority traffic classes but employed a simple on/off switch to tell the routing layer that an latency-sensitive application is active and currently sending high-priority traffic.

5.2.3. Multi-hop Simulation

We evaluated the previously mentioned features in NS-2 with 20 nodes and standard settings for the wireless network. Thus, the transmission speed for all packets was fixed at 2 MBit/s. All nodes moved according to the random waypoint mobility model with speeds between zero and three metres per second and a pause time of 180 s. The simulation area was $650 \times 650\,m^2$ and the simulation lasted ten minutes per scenario. All scenarios were repeated ten times using five flows of low and three flows of high priority traffic with 20 packets per second and a fixed packet size of 64 byte. With this traffic pattern, our evaluation is not congested but still under considerable load. The aim of this evaluation was to discover what effect the previously mentioned features have on high priority traffic. We then compare our results with network requirements for interactive latency-sensitive applications to determine the maximum number of hops we can achieve in a MANET.

Figures 5.6 and 5.7 show the results of our evaluation. In both figures we start on the left by showing the results of a default NS-2 configuration with RTS/CTS enabled but with no additional QoS feature. We then start to add one feature after the other, showing a combining impact so that the rightmost bar shows a scenario with all QoS features enabled. We decided that the application in our example drops all traffic that exceeds a one-way delay of 100 ms to emulate that this data is not useful to interactive applications anymore. Hence, packets with high delay are shown as packet loss in Figure 5.7 and the average latency in Figure 5.6 is calculated for the remaining packets only. We previously discussed that we assume a round-trip time limit of 150 ms for interactive applications. We deliberately chose to exceed this value in our simulation and use a higher one-way latency of 100 ms to make sure we definitely exceed this value in the case of asymmetric latencies. As we can see from Figure 5.6 delay for the remaining packets that are successfully delivered to the application are relatively low and vary between six and eight milliseconds delay per hop. Obviously as the number

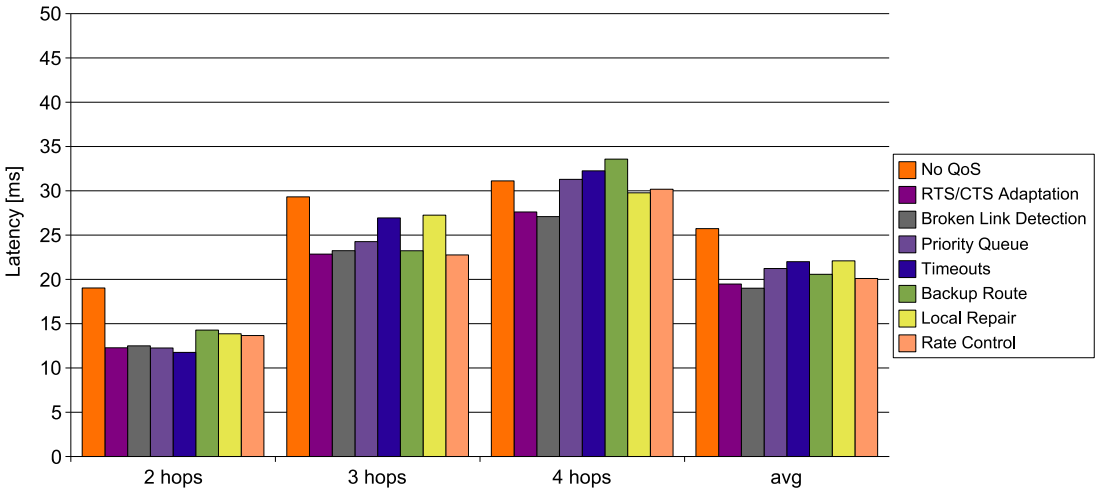


Figure 5.6.: Average one-way latency for high priority traffic (from [160])

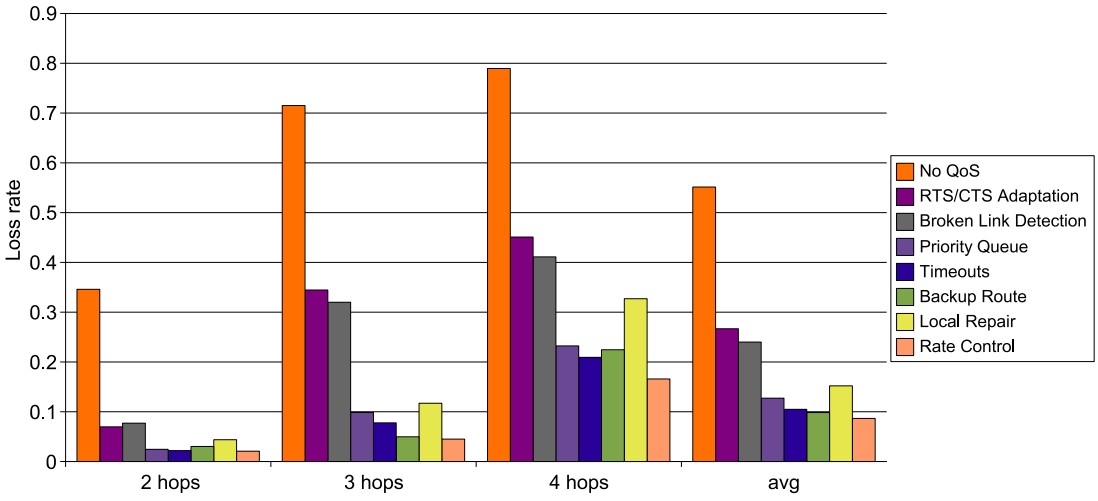


Figure 5.7.: Average packet loss for high priority traffic (from [160])

of hops increases, the overall end-to-end latency as well as the per hop delay goes up as well due to the additional transmission and congestion overhead at intermediate nodes. Generally, without regarding any high-delay packets, we achieved a one-way delay of approximately 30 ms over four hops. Thus we can conclude that individual data packets are either well within the delay requirements of interactive latency-sensitive applications or they exceed that limit. Figure 5.7 shows the percentage of packets that have a delay of more then 100 ms or that are lost during transmission. As before, we assume that interactive applications can cope with a maximum of five percent packet loss. In our scenario with the default NS-2 settings, even communication over two hops exceeds this value dramatically. For communication over four hops, we show nearly 80 percent packets that are either lost or exceed the required delay with some margin. Using RTS/CTS adaptation helps to reduce packet loss, most likely because of the reduced overhead if no RTS/CTS frames are sent. Although such packets are very small so is the payload of our interactive data traffic which makes RTS/CTS overhead noticeable. Also, the lack of a mechanism against hidden terminals does not seem to have a negative effect on data transmission in our scenario. The next noticeable drop in packet loss comes with the introduction of priority queueing which seems to work well even without any support from WLAN's distributed coordination function. It is noticeable that enabling the local repair feature actually increases packet loss and to a lesser extent delay. The reason for this is that the repaired route is often longer than the original one. In this case, local repair automatically triggers a route error notification message back to the sender which then restarts the route discovery process. As a result, local repair is often ineffective and leads to increased overhead.

For background traffic, our QoS features had the expected negative effects on latency which was more than doubled with all features enabled. However, packet loss for low priority traffic stayed at about the same level when comparing simulation runs with no and with all features active. The detailed results for the background traffic as well as the results for each individual QoS features can be found in Appendix A.3.

5.2.4. Conclusions

From our simulation results we can see that it is not possible to fulfill the network requirements of interactive latency-sensitive applications of less than 150 ms round trip time and less than five percent packet loss in our chosen scenario because too much packets were either lost or experienced single way latencies above 100 ms. One reason for this is the high network load and low transmission speed combination we have chosen for our scenario. Other reasons come from the characteristics of WLAN multi-hop communication. If an intermediate node forwards data, it blocks transmissions of all

other nodes in its vicinity. Thus if the queuing delay of a packet at the intermediate node is greater then the sending interval of the application, in our case 50 ms, the transmission of the following packet to the intermediate node is affected by the forwarding transmission of the initial packet from the intermediate node. Also, the interference range of a wireless radio is larger than its communication range.

We have also shown that with the introduction of several simple QoS features, we were able to reduce delay and packet loss so that on average the network requirements for interactive applications can be met up to a distance of three hops. RTS/CTS adaptation and priority queuing and to a lesser extent rate control were the most effective methods to increase network quality for high priority data traffic.

From the simulation results, we can draw some conclusions regarding our server selection algorithm. In the previous chapter, we discussed that we designed the algorithm to discover servers at distances of one or two hops. With our simulations indicating that a maximum of three hops are suitable for delay-sensitive traffic, this means that a mobile node can move away from it's server and still maintain a viable connection to it. Moreover, if we compare the differences in latency and packet loss between two, three and four hops in Figures 5.6 and 5.7, we notice that changes in both values are quite large if the number of hops changes. Thus, it may be prudent for applications to start looking for a new server when the distance between client and server reaches three hops.

5.3. Chapter Summary

In this chapter we discussed the network performance of interactive latency-sensitive applications in wireless ad-hoc networks and proposed protocols and methods so that their requirements can be achieved. We started by measuring single-hop wireless ad-hoc communication over various distances and background traffic conditions as well as simulated different ad-hoc routing protocols. We then moved on to look at a multi-hop scenario and discussed various optimisations aimed at improving the quality of service for our applications.

We have shown that wireless LAN is generally suitable to fulfill the network requirements of latency-sensitive applications. We argued that the maximum number of low-layer retransmissions should be reduced to six after which the successful delivery of a data packet becomes unlikely. Furthermore, we discussed the existence of a rate-dependent hidden terminal problem that effects communication at larger distances when background traffic is being sent by a remote node.

The routing procotol AODV showed the best results in our initial NS-2 simulation although it did not fulfill all necessary application requirements. For the simulation of a multi-hop network, we presented several QoS approaches to improve network performance. Finally, we have shown that with these modifications a maximum of three hops between client and server can be achieved within the requirements of latency-sensitive applications.

In the previous chapter, we discussed the server selection algorithm which can determine servers at a maximum distance of two hops. We have chosen this limit intentionally because at this distance network performance is usually good for latency-sensitive applications. Furthermore, if nodes move and the route is extended to three hops, communication still remains within the necessary limits. Then, mobile nodes have the necessary time to choose an alternative server and complete the transition before communication degrades further.

6. Evaluation

Our evaluation combines several points already discussed in previous chapters. Here, we analyse the performance and scalability of the server selection algorithm in various scenarios. We also use the results as well as lessons learned from our preliminary evaluation and theoretical analysis to explain our evaluation results. We aim at showing how well our server selection algorithm can perform under good conditions, explain its fair performance under challenging conditions and show its limits. Also, we take a close look what effect a server selection has on the network traffic of a latency-sensitive application.

This chapter is divided into two parts. We start by looking at different evaluation methods to find the best way to evaluate the performance of the server selection algorithm and its subsequent effect on the performance of a latency-sensitive application. We then describe how to use real world measurements to select and improve an existing network simulator as a suitable MANET evaluation tool for latency and packet loss. Moreover, we describe the network communication for the multi-player game CounterStrike which we use as our example of an interactive and latency-sensitive application in our evaluation. We conclude the first part with details regarding our node colouring-based evaluation which we use to present our evaluation results.

In the second part we present and analyse data from our evaluation. We start by looking at two artificial scenarios to determine general network overhead and scalability of the server selection algorithm. We then continue to look at the algorithm and application performance in two real-world scenarios, a schoolyard and a train ride. Moreover, we extend the schoolyard scenario with mobile game nodes to study the effect of mobility on our results. Finally, a summary concludes this chapter.

6.1. Methods & Tools

Algorithms for mobile ad-hoc networks can be evaluated through various means. Basically, these evaluation methods can be divided into three categories[178]: Measurements or field testbeds, emulation, and simulation. With measurements the algorithms

run on mobile devices in the specific real-world scenario. Measurements need a lot of hardware and manpower in order to arrange a test run but they also provide the most realistic results for a scenario. The main disadvantage of measurements is that they are hard to reproduce. A changing environment, like people or cars moving differently than in the previous test, can have a huge impact on the performance of the network and the results of the test. Hence with measurements, it is hard to tell whether a better performance comes from a better version of the algorithm or a changed environment

Secondly, network emulation tries to bypass the problem of reproducibility in a changing environment by imitating the network. Here, real devices are wired directly with a network emulation computer and often Ethernet is used to replace the wireless links. The emulation computer then recreates predetermined network characteristics in order to create a suitable testing environment. Because the network is only emulated, the same scenario can be tested over and over again. One disadvantage of emulation is that accurate timing of data signals would need to be in the order of several microseconds which can be difficult to achieve by the emulation computer. Therefore, network emulation often operates on a higher network layer that influences throughput, delay, delay variation and packet loss through various statistical functions. In advanced emulations, the antenna sockets of several devices are connected directly by cables without the need for any central emulation system. This approach can improve realism by using the real wireless protocol and solves the timing issues. Still, additional effort is needed to emulate node mobility, signal degradation and multiple collision domains. Both measurement and emulation require the provision of real devices to improve realism but which does not scale very well.

Finally, network simulation artificially recreates mobile devices themselves as well as the wireless network. Because all parts of a test run are embedded inside a simulation, it is able to produce an optimal time resolution because of it's ability to run the simulation slower than real-time. On the other hand, simulations need to reproduce all aspects of hard- and software from top to bottom in order to be fully realistic. However and due to reasons of practicability, in most cases only necessary and important features are recreated.

We believe that extra care should be taken when evaluating algorithms for mobile ad-hoc networks as each presented method has its distinct set of advantages and shortcomings. Table 6.1.1 shows a comparison of all three methods. For the evaluation of our server selection algorithm reproducible results are essential because they allow us to compare the results of different scenarios and setups with each other. Therefore, we need to use a setup such that we are able to attribute any changes in the behaviour of our algorithm exclusively to the modification in the setup of the simulator rather than to outside interference. Additionally, the exact timing of data packets for latency sen-

	Practicability		Realism	
Method	Effort	Reproducible	Timing	Real hardware
Measurement	high	–	+	+
Emulation	medium	+	–	o
Simulation	low	+	+	–

Table 6.1.: Comparison of different evaluation methods

sitive applications is of importance. Consequently, we have decided to primarily focus on simulation for our evaluation. However, we have done additional measurements which we use to verify the eligibility of our simulation tool.

6.1.1. Simulator Selection

Throughout the research community, several different simulation tools for wireless networks are being used, the most important ones being NS-2, GloMoSim, QualNet, and OPNET. According to [179], NS-2 is the most prominent simulator and widely accepted in the research community therefore we will also use it for our evaluation.

NS-2[152]is a discrete event simulator that has been developed over the last nearly twenty years. Although NS-2 is now being hosted by the Information Science Institute at the University of Southern California, it includes substantial contributions from other institutions and researchers and was partly financed by DARPA, NSF and others. The major part of the wireless and mobility code in NS-2 comes from the Monarch project[180] at Carnegie Mellon University. CMU's wireless system was added in 1999 and includes a two ray ground propagation model, the implementation of a Lucent WaveLAN DSSS radio and the IEEE 802.11 DCF MAC protocol as well as several multi-hop ad hoc network routing protocols. Despite the fact that NS-2 is widely used for simulating WLAN networks, many researchers find it disappointing because, among other things, it does not support beacons, management frames or multi-rate data transmissions and uses only a simple power-based interference model which is calculated between two packets.

Because of these shortcomings and further advancements in WLAN technology since 1999, we looked at alternative implementations of wireless components for NS-2 to see if they are more complete, more realistic and more suited to our evaluation needs. Especially, we were looking for a multi-rate implementation of IEEE 802.11a or .11g which would make delay simulations more accurate and for a better interfer-

ence model which would calculate packet loss on the wireless link more precisely. The IEEE 802.11 implementation[181] of Mathieu Lacage from INRIA which was developed in 2005 fulfills all these requirements. It replaces the NS-2's WLAN stack with an implementation of 802.11a, .11b, and .11e HCCA and EDCA. It supports all data rates available in 802.11a and implements both the ARF and the AARF rate control algorithms. Like the standard NS-2 implementation it provides a simple energy-based interference model. Moreover, the INRIA stack implements two additional models: a signal-to-noise ratio threshold based model and a bit-error rate based model. The latter of which calculates the bit-error rate dependent on the modulation of the transmission. An overview of other NS-2 related WLAN implementations and patches can be found in [182].

Table 6.2 shows the difference between the original and INRIA's implementation of IEEE 802.11 protocols for NS-2. We divided this comparison into three parts: Radio technology, Radio models and the Modelling of the Environment. In the real world, you can find three different WLAN variants, namely .11a, .11.b, and .11.g. Nowadays, most devices use a combination of .11b and the newer .11g variants with a gross transmission speed of up to 54 MBit/s. Such devices would usually use the .11g high-speed variant, but these devices would fall back to the more robust .11b in cases where interference is high. In comparison, NS-2's standard WLAN stack only supports 802.11b with speeds up to 11 MBit/s. Additionally, the transmission speed for each node has to be set to a fixed value by the researcher for the duration of a simulation run. INRIA's stack allows for high-speed WLANs and implements data rate adaptation techniques that allow to dynamically change the transmission rates to the maximum value that can be sustained between two stations. But because of its implementation of the 802.11a standard, INRIA's stack cannot fall-back to transmission rates lower than 6 MBit/s that are often used with current mobile devices under difficult conditions. Finally, properties like multipath propagation, fading or non-line-of-sight communication (NLOS) are not implemented by either stack.

6.1.2. Simulator Shortcomings & Adjustments

During our initial tests with both WLAN stacks in NS-2, we identified several problems when simulating wireless networks with NS-2. First and foremost was the lack of ARP request retries. When a real IP unicast transmission between two stations is about to take place, the lower layer of the sender usually sends out an ARP request to determine the MAC address of the receiving node. If there is no response to that request, the ARP request is sent several times before an error is reported back to the higher layers. However, NS-2 does not retransmit ARP requests. Because of movement and the fact

		Real world	NS-2 Std.	NS-2 INRIA
Radio Technology	802.11	a/b/g	b	a
	Rate Adaptation.	Various	N/A	ARF, AARF
	Retries	Multi-Rate	Same Rate	Same Rate
Radio model	Propagation	Realistic	FreeSpace TwoRay Shadowing	FreeSpace *Shadowing*
	Multipath	Yes	No	No
	Fading	Yes	No	No
	Reception	Realistic	SNRT	Energy, SNRT BER
	Noise	Realistic	Compare 2 transmissions	Thermal noise + oth. signals
Modelling of Environment	Obstacles	LOS/NLOS	LOS only	LOS only

Table 6.2.: Comparison of different WLAN implementations for NS-2 and the reality

that frames in wireless networks get lost more frequently than in wired networks, this behaviour is problematic because without a valid link layer address data packets will be discarded on the sending node. Furthermore, any data retransmission method on the lower layers will not be used because the packet itself is removed from the queue at layer 3. For TCP streams this often prevents the establishment of a connection in general. Because of its importance we implemented retransmissions of ARP requests in NS-2. A maximum of four ARP requests, the initial request plus three retransmissions, are sent for every station before an error is reported back to the higher layer which is the current behaviour of the Linux 2.6 IP network stack.

Another problem concerns broadcast packets. With WLAN, broadcast packets need to be handled differently than unicast packets. In contrast to unicast packets, broadcast packets are not acknowledged on layer 2 because there is no single receiver that could acknowledge the packet. Because of this, broadcast packets are usually sent at low speed with robust coding to ensure successful reception. The WLAN standard defines two transmission rate sets for wireless devices: the basic rate set and the extended rate set. Support for data transmission rates in the basic rate set are mandatory for all devices while the implementation of the usually higher transmission rates in the extended rate set is optional. Broadcasts must be sent at a transmission rate that is contained in the basic rate set so that all nodes can receive them. For 802.11b the

basic rate set contains the 1 MBit/s and 2 MBit/s transmission rate. The INRIA stack uses adaptive rates that is controlled through a rate adaptation algorithm as in real devices. The standard NS-2 WLAN stack sends all packets at fixed transmission rates so that either broadcasts are sent with higher than usual transmission rates or data frames are sent at lower speeds. The behaviour distorts not only the network impact on the data frame in question but also influences how long a node occupies the wireless channel. Furthermore, longer or shorter transmissions can have an impact on whether data sent from other nodes is lost due to interference by this transmission. Because we use broadcast packets extensively for neighbour detection in the server selection algorithm, we enhanced NS-2 by implementing a restriction to the basic rate set for broadcasts.

We also added low-layer WLAN ad-hoc beacons. For this, every station that participates in the ad-hoc network operates a timer that schedules the transmission of a WLAN beacon every 100 ms. This WLAN beacon announces the availability of the ad-hoc network to all other nodes and is sent as a broadcast. If a station receives a beacon for its ad-hoc network, the timer is reset to ensure that only one station transmits that beacon in a certain area of the network. To minimize collisions when beacon timers of multiple nodes fire at once, a small random number is added or subtracted from 100 ms each time the timer is started.

The INRIA stack also has some shortcomings in terms of features regarding the evaluation of mobile ad-hoc networks. Hence, we ported some useful features from the standard NS-2 system to the INRIA stack. These features include general node mobility support, the AODV routing protocol and the shadowing propagation model.

Finally, we adjusted some low level parameters of the INRIA stack to resemble the characteristics of an Atheros-based WLAN card under Linux which we used in our real-world measurements in Chapter 5. For this, we extended the data reception mechanism to allow for different RX thresholds for different data rates. With real hardware, high transmission rates require a higher signal-to-noise ratio to be received successfully than lower rates. We also adjusted several other parameters such as the transmission power of the radio and the maximum number of unicast packet retries in accordance with documentation available from Atheros Inc. and the Linux kernel driver.

6.1.3. WLAN Stack Comparison

In order to choose the most suitable and realistic WLAN stack for the evaluation of our server selection algorithm, we compared simulated results with measurements from a small real-world setup. Besides choosing the best WLAN stack, we also wanted to

see how accurate the results from the NS-2 simulator are when compared to real-world measurements. We used three notebooks fitted with Atheros PCMCIA WLAN cards and measured the traffic between two notebooks at various distances. The third notebook was used to create background traffic. This is the same scenario and the same measurements that we used in Chapter 5. You can find a more detailed description of the setup there. For our comparison, we focus on three network parameters, namely delay, delay variation and loss because these are the most important network properties for latency-sensitive applications. We measured at distances between two stations of 1, 5, 25, and 50 meters each with and without background traffic for application packet sizes of 104, 576 and 1400 bytes.

We also repeated the measurements three times to rule out any temporary interference at the time of measurement. We then reconstructed each scenario in the computer and simulated it with NS-2's standard stack and INRIA's stack. In order to allow for a fair comparison, we run the two different tests with the standard stack, one with the transmission rate set fixed to 11 MBit/s and one with an artificial fixed 54 MBit/s setting although the implemented 11.b does not originally allow for such a high speed transmission. We used shadowing as the propagation model in our simulations which is defined as follows:

$$\left[\frac{P_r(d)}{P_r(d_0)}\right]_{dB} = -10\ \beta\ log(\frac{d}{d_0}) + X_{dB} \tag{6.1}$$

The formula comprises two parts, the path loss model and the shadowing variation. The path loss model calculates the signal strength at the sender at distance d in reference to a close-in distance d_0. The path loss can be adjusted through the path loss exponent β to reflect different specific indoor or outdoor scenarios. The shadowing variation extends the ideal circle model of radio propagation by introducing an additional probabilistic component which influences signal reception near the edge of the communication range. Hence, a Gauss-distributed random variable X with zero mean and standard deviation σ_{dB} is added to the calculated path loss. The parameter β and σ_{dB} are empirically determined. Typical values for both parameters can be found in [183]. More information about the shadowing propagation model can also be found in [184]. In the literature, we can find a wide range of values for both the path loss β and the shadowing deviation σ for an outdoor urban area. Therefore, in our WLAN stack comparison, we run each simulation multiple times with different values for both parameters which covers the whole range of the parameter set that is shown in Table 6.3.

In order to be able to evaluate solely the behaviour of the WLAN stack, we specifically decided against measurements of multi-hop ad-hoc traffic as not to be influenced by ad-hoc routing or other effects that may occur on higher layers. As mentioned, we

Variable	Value	Interval
Path loss exponent β	2.7 – 5.0	0.1
Shadowing deviation σ_{dB}	4 – 12 dB	1 dB

Table 6.3.: Parameters for the shadowing propagation model for outdoor urban areas (based on [183])

Value	Measurement	NS-2 11	NS-2 54	INRIA
Avg. RTT	9.1 ms	13.3 ms	7.99 ms	8,7 ms
Difference		+4.2 ms	+1.1 ms	-0.4 ms
Avg. RTT variation	7,0 ms	2.1 ms	1.66 ms	2.1 ms
Difference		-4.9 ms	-5.3 ms	-4.9 ms
Avg. Loss	0,7 %	3.3 %	49.8 %	0.0 %
Difference		+2.6 %	+49.1 %	-0,7%

Table 6.4.: Best match results from the WLAN stack comparison

evaluated a variety of settings for the path loss β and the shadowing deviation σ. For each WLAN stack we selected the evaluation result that matches the measurement best. Thus, we picked only the best-case results to see which stack comes closest to our real-world measurement. Table 6.4 shows the results of our comparison for a single scenario. Here, the two measured nodes were 25 meters apart and used a packet size of 104 bytes while at the same time a third node transmitted full speed TCP background traffic. Although these numbers only represent a single one out of 24 different scenario, the results are typical for our WLAN stack evaluation.

From Table 6.4 we can see that the INRIA WLAN stack offers the best results. It is able to match the measured delay quite well, however like the NS-2 original stack it fails to achieve a good delay variation. For packet loss, the results look quite reasonable for all stacks but for NS-2 54 MBit. This stack suffers greatly from its inability to change its data transmission rate during phases of high load which is produced by our background traffic. From these results we conclude that the INRIA stack is the best choice for our evaluation. However, the quality of the simulated results regarding RTT variation is low with all stacks which may be the result of a missing signal fading model. However, a full evaluation of this matter is out of scope of this thesis.

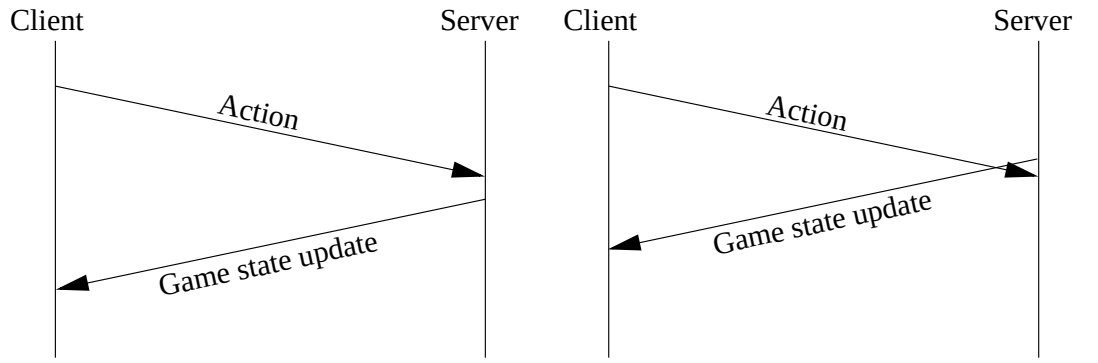
	Model by Faerber		Evaluation	
	Server (per client)	Client	Server (per client)	Client
avg. inter-arrival time	62 ms	41.7 ms	60 ms	40 ms
distribution	extreme	deterministic	deterministic ±5%	deterministic ±5%
avg. packet size	127 bytes	82 bytes	127 bytes	82 bytes
distribution	extreme	extreme	deterministic	deterministic
protocol	UDP			

Table 6.5.: Traffic characteristics of the multi-player game '*Counterstrike*' (from [8])

6.1.4. Traffic characteristics and measurement methods

We have modelled the application traffic according to the action multi-player game Counterstrike which besides its age remains quite popular among players of this genre. In [8], the author analysed the game traffic of Counterstrike in detail and created a model for client → server and server → client game traffic. We have mainly relied on this traffic model with the exception that for ease of use we use fixed packet sizes and rounded the packet inter-arrival times both of which are shown in Table 6.5. To prevent deterministic behaviour where multiple nodes regularly transmit at the same time, we added a random uniformly distributed offset of ±5 % of the original value to the packet inter-arrival time.

When a client has discovered its server through the server selection algorithm, it starts sending data packets of 82 bytes to its server. This data packet contains requested actions from the user which are sent to the server for inclusion in the next game state. The server collects all such actions from its clients, compiles a new game state approximately every 60 milliseconds and sends an update of the game state to its clients. With this update, the client then can see whether its action has been approved or rejected by the server. Thus, to measure the delay of any game action we have to look at the round-trip time from when the client sends its request until it receives the response from its server. Because our game traffic is asynchronous and does not follow a simple request / response policy, we cannot simply calculate the round-trip time based on the data packets sent back and forth. Figure 6.1 shows an example where a client sends an action request to its server and receives an update shortly after that. The client now cannot decide if its request has been received by its server in time to be incorporated into this response or if it had come too late and will be acknowledged by the next up-



(a) Update includes action from the client (b) Update does not include action from the client

Figure 6.1.: Effects of delay on game traffic

date. Thus, we define an action as an activity of a user which leads to the transmission of a game packet to the server. We have implemented the handling of actions as follows: Each action carries a client-specific ascending ID which uniquely identifies that specific action. Upon reception of an action, the server stores the action ID in a per-client list. When the next game state update is due, the server adds a list of remaining action IDs as acknowledgements to the packet and clears its list. The client can now calculate the round-trip time of an action by looking at the transmission time of the action and the time of the reception of its acknowledgement. We define this round-trip time as the *action delay* for a specific action.

Figure 6.2 shows an example of two clients communicating with one server. It shows that because of the different packet inter-arrival times for the clients and the server, the server has sometimes to include multiple actions in a single game state update message. It can also happen that no action was received by the server thus the game state update contains only actions from other clients. Of course, in the case of packet loss, e.g. for action no. 3 of client 1, the server is not able to consider this action in its subsequent game update message and it will be lost.

For the reasons mentioned above, we base the following evaluation on the action delay, action delay variation and action loss rather than to rely on packets or frames. By focusing mainly on actions for our evaluation, we take the application specific behaviour into account which closely resembles the application behaviour as seen by the user. Hence, we believe this approach to be more realistic than simple packet- or frame-based evaluations.

However, one disadvantage of this approach should also be mentioned. In case of lost game state updates, the client does not receive an acknowledgement for its action and

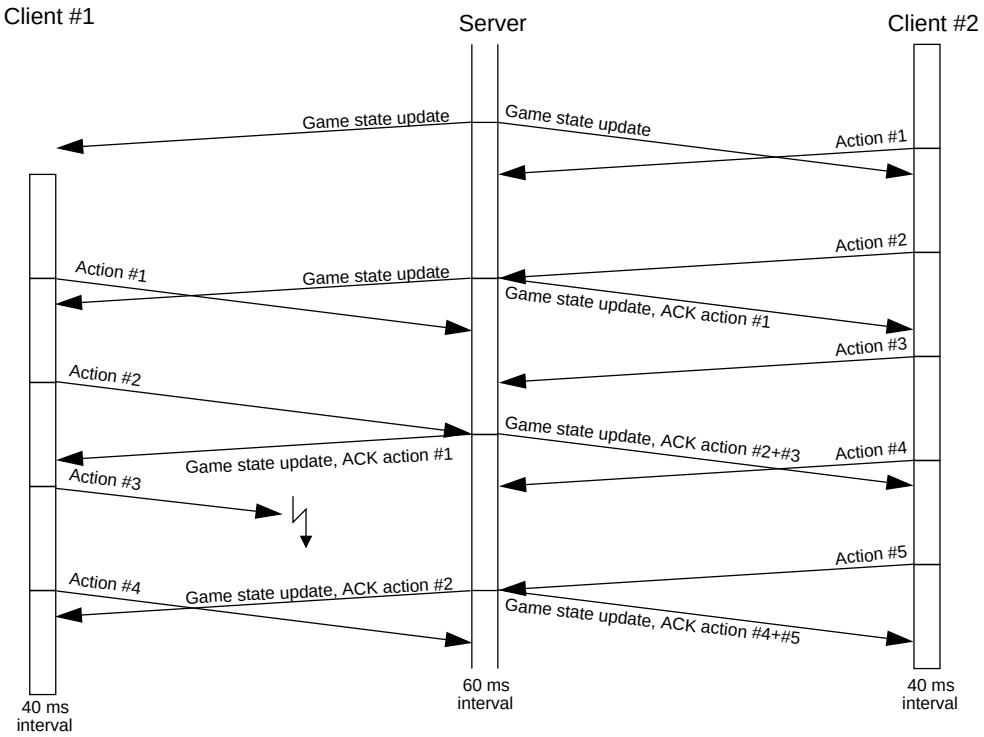


Figure 6.2.: Game traffic communication example

presumes that its action has been lost during transmission as both cases are indistinguishable from the client side. Hence, no action delay can be calculated. In reality, if the action was received by the server it has already been included in the calculation of the next game state. Thus, the following game state update that is received by the client would include its previous action in a real game. We did not implement this behaviour and used a simplified game state model for our evaluation. As a consequence, our action loss ratio may be slightly higher. However, we believe that the influence of this behaviour has little consequences and would be present in a packet- or frame-based evaluation as well.

In addition to the game traffic between clients and their servers, the zone servers need to synchronize their game states with each other to provide all players with a consistent view on the game. As previously discussed in Chapter 3 game server synchronisation depends on the actual game play, the implemented game architecture and algorithms. Therefore, we have implemented a small and simple synchronisation scheme between servers which uses the traffic characteristic from game traffic sent by one client to a server. Although the total throughput between two servers is lower than the data sent between a game server and its several clients, we believe that data aggregation

and delaying techniques will be used widely between servers to minimize the load of the mobile ad-hoc network between servers. Finally and in addition to the game traffic, we add background traffic between supporting nodes in the network in some scenarios. For each scenario, we define a background traffic probability and check for every supporting node whether it is a background sender by using a uniformly distributed random number. For each sender, we then choose a background receiver from the group of supporting nodes and make sure that sender and receiver are different nodes. Like in reality, it can happen by chance that two or more nodes send traffic to the same background receiver.

Node colours and states

During a scenario run, game nodes can encounter various problems. Their game traffic can be delayed or lost or their game server may become unavailable. Furthermore, new games servers can be created at any time if the situation warrants it while existing servers may drop their server role because they do not have clients anymore. To better understand what happens in a scenario, we give a colour to each node depending on its current application status at any given time during the simulation. We define seven different colors which are shown in Figure 6.3. The first two colours are used mainly

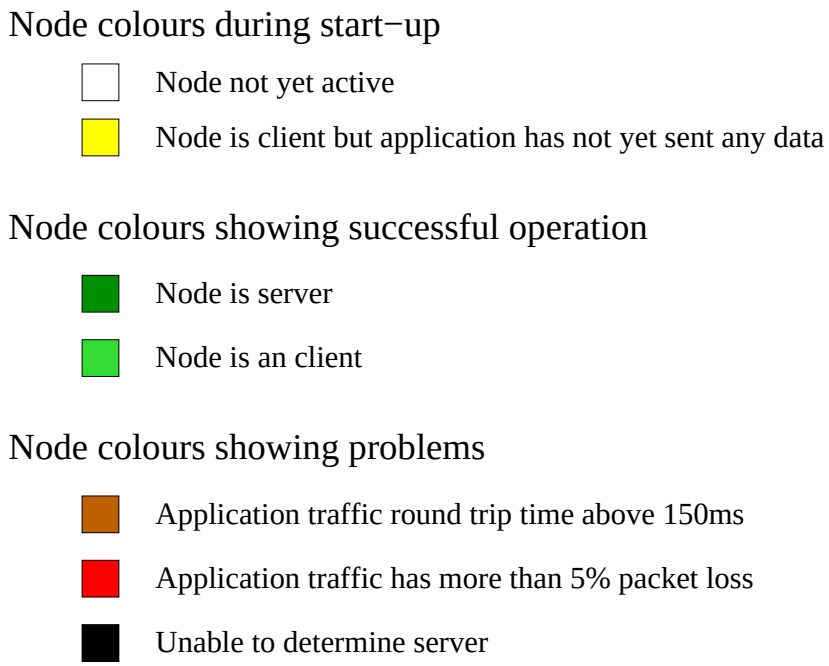


Figure 6.3.: Game node colours which show current application status

at the beginning of the scenario. All nodes are given the colour white at the beginning of a simulation run. They retain this colour during the thirty seconds warm-up phase for the AODV routing algorithm. After a node has started, it changes its colour to black because the server selection algorithm has not yet found a solution. This is a normal situation because the server selection algorithm has just started to operate on this node. After a solution has been found, our node changes its colour either to dark green to show that it operates as a server or to yellow which means that the node is a client and has successfully found a server nearby.

In order to determine the status of the application, we record end-to-end delays for data sent to and received from the server. We then continuously determine the average round trip time for the application by using a sliding window with 25 actions. Until our sliding window has collected network performance data from 25 actions, the colour of a node remains yellow. After that, the colour can change to light green which indicates that everything is working fine. If the average round trip time increases above 150 ms the colour changes to orange and if more than 5 % of actions got lost then it will change to red. In the case both values exceed their performance limit, the node's colour also changes to red as to indicate a situation where the application is not working properly.

While node colours are a good way to visualise the current status of an individual node, they cannot describe the result of a scenario as a whole and be used to directly compare different scenarios or simulation runs. For this, another metric is required which can describe the quality of the application during the simulation. We introduce four different states that describe the status of the application:

- Playable – green and dark green colours
- Non-playable – orange and red colours
- Lost / no server – black colour
- Setup and Determination – white and yellow colours

The playable state comprises both green states where the application either has a server nearby or the local node is a server itself. In both cases the round-trip time and the packet loss are within acceptable limits of less than 150 ms and 5 % respectively. The non-playable state indicates that the application cannot work properly as traffic limits are not met and that either the round-trip time (orange colour) or the packet loss (red colour) are above the given threshold for our application. In both the playable as well as the non-playable state, the server selection algorithm has successfully determined a server. The lost state shows that the application is not running because the server selection algorithm was unable to determine a server. It is also used briefly during the startup period of a node when the node has not yet determined a server. Finally, all the

remaining node colours are put into the setup and determination state.

To be able to compare different scenarios we define the term 'playability time' in which we add up all times of all nodes in a simulation run where the application is in the playable state. We then divide this sum by the total duration of the simulation run. Thus, the playability time defines the fraction of simulation time where the game is playable. The same can be done for the other three states as well.

6.2. Evaluation Results

In Chapter 2 we introduced three different scenarios for mobile applications. Below we will present the results and the conclusions from our simulations in the schoolyard and the train ride scenario. Both scenarios have in common that background nodes move around freely in the designated scenario area and that application nodes do not move at all. The reason for this as discussed in Chapter 2 is that the user needs to focus his attention on the application in order to use it. Additionally, we decided against using the train station scenario because of its similarity to the school yard scenario. Instead we will use a supplemental artificial scenario that will show how our algorithms behaves when all nodes are moving. Moreover, we will discuss the simulation results regarding scalability of the server selection algorithm and draw our conclusions.

Table 6.6 shows the general and node-specific parameter settings for each scenario. Because each individual scenario has its distinct environment, we use different parameters for the shadowing propagation algorithm that are adapted to suit each specific scenario. We repeated every simulation run 20 times with different seeds for the random number generator. We also made sure that we created random numbers properly as described in [185].

6.2.1. Algorithm Overhead and Scalability

Before we evaluate our server selection algorithm in realistic scenarios, we take a look at the scalability of the algorithm. We start by looking at how long the algorithm takes to find a solution for a node from the time the algorithm starts until it has found a nearby server or determines that the node should be a server itself. Secondly, we take a look at how much data traffic our algorithm sends and how much it receives from other nodes. We define the network overhead as the sum of sent and received traffic because in both cases the wireless media is busy and the node is not able to receive

Variable	Values	Number of simulations
General settings		
Path loss	scenario specific	
Shadowing deviation	scenario specific	
No. of random number seeds	20	20 initial simulations
Node settings		
No. of application nodes	5, 10, 15, 20, 25	5 different values
No. of background nodes	0, 5, 10	3 different values
Total number of nodes	15 – 35	
Background traffic probability	0, 0.5, 1	3 different values
Simulation runs per scenario		900 simulations

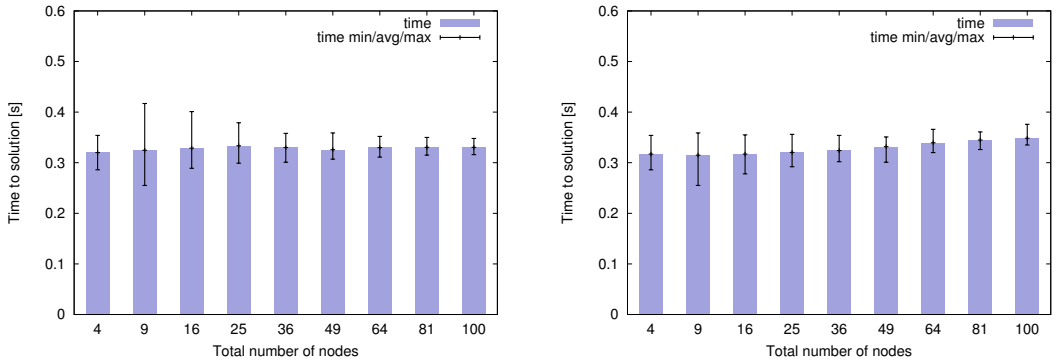
Table 6.6.: Simulation parameters for the evaluation

or transmit application data. Because the server selection algorithm sends locally and uses information from neighbouring nodes only, we expect a good performance with an increasing number of nodes.

We have chosen two different scenarios for our scalability simulation. The first scenario is a one dimensional scenario where we form a line of nodes. The distance between two nodes is one metre. In the second scenario, we use a two-dimensional rectangle with a distance of one meter vertically and horizontally between adjacent nodes. In order to form a full rectangle we increase the number of nodes quadratically up to 100 nodes. We have simulate each scenario 50 times with a simulation time of 90 seconds.

Figures 6.4 shows the average server determination time in seconds in both scenarios. The height of the bars displays the average time for the server selection algorithm to find a solution over all 50 repetitions of the same scenario. We also show the minimum as well as the maximum time for a given number of nodes. In both figures the time on the y-axis is in linear scale while the number of nodes on the x-axis increases quadratically. We notice that the average determination time remains around 330 milliseconds for the linear scenario. In the rectangular scenario we see a similar behaviour only with a slight increase up to 350 ms for an increasing number of nodes. Still, both graphs show that our server selection algorithm can deal with an increasing number of nodes almost in constant time.

Another important factor for the scalability of a distributed algorithm is its communication overhead. This overhead usually includes the number of bytes that a node



(a) Nodes in a line

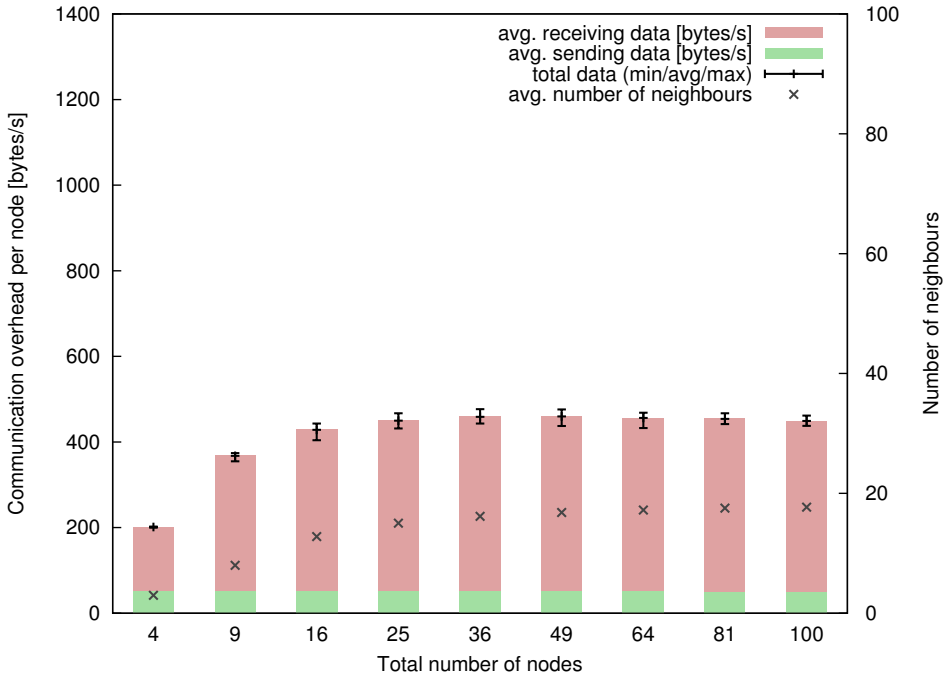
(b) Nodes in a rectangle

Figure 6.4.: Average server determination time

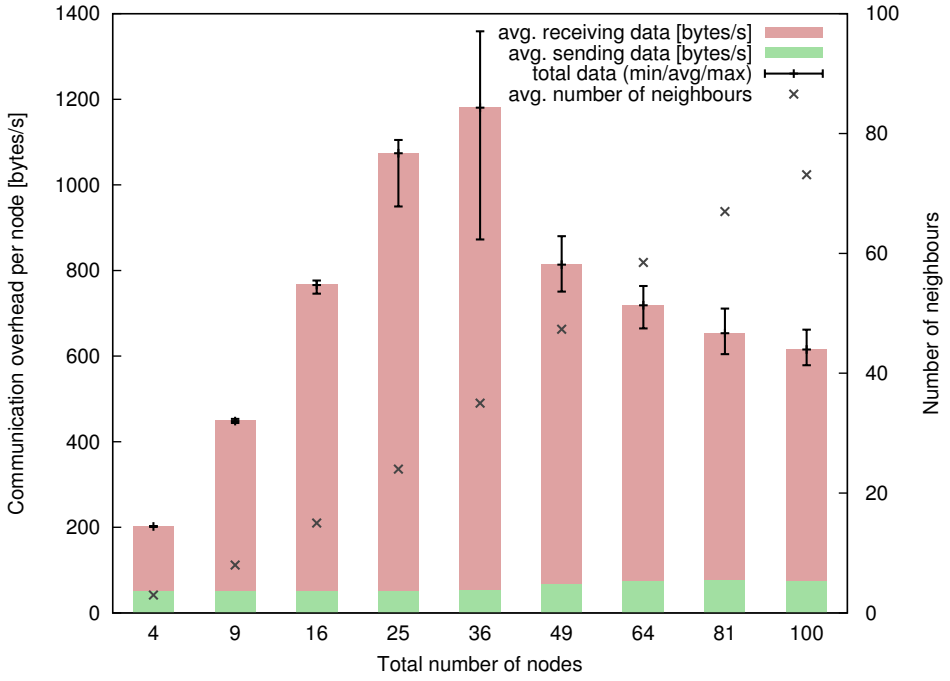
sends into the network. In our analysis, we decided to go a slightly different way and include traffic that is received at a node as well. The reason for this is twofold. First of all, wireless networks such as IEEE 802.11abg can operate in simplex mode only. That means that data transmission and reception can only be done exclusively because data is transmitted and received over the same channel. Therefore, if a large number of packets is received at a node, it prevents the node from sending its own data. Secondly, the wireless medium is a scarce resource and has a smaller capacity than wired networks. As a result and in combination with the simplex mode, the transmission as well as the reception of data have a direct impact on latency for interactive applications.

Figure 6.5 shows the communication overhead in bytes/s per node both scenarios. The green bar shows the number of bytes that a single node sends on average per second and the red bar shows what it receives. Again, the black error line shows the total communication overhead on average as well as the maximum and minimum values across all 50 runs of a scenario. Additionally, we present the average number of neighbours as seen by the server selection algorithm in each scenario. In both figures we notice that a node receives far more traffic from other nodes than it sends itself. This is easy to explain as in both scenarios the average number of neighbours exceeds one considerably. Secondly, we can see that a node sends about the same amount of data regardless the total number of nodes in the scenario which is the result of using broadcasts. However, there is a slight increase in the number of bytes a node sends in the rectangular scenario as the number of nodes grow. This increase is caused by retransmissions which we will discuss in more detail in the following paragraph.

We start by looking at Figure 6.5(b) on the right side which shows the overhead for the rectangular scenario. We can see an almost linear increase in traffic up to a total



(a) Nodes in a line



(b) Nodes in a rectangle

Figure 6.5.: Communication overhead per node

node number of 36 after which the traffic decreases again. One reason for that could be that with growing numbers of nodes and the total size of the network increasing as well, it gets too large for single-hop data, in our case broadcast frames, to reach every node in the network. However, with 49 nodes and a maximum node-to-node distance of less than 10 meters, this reasons seems unlikely. Also, we notice a linear increase in neighbouring nodes in the same figure. In fact, the real rationale for the drop in received data traffic are collisions that occur in the network. To further underline this statement, we take a look at the average number of tickets send per node as shown in Figure 6.6. Tickets are sent by a node when its server selection algorithm has determined a prospective node as a suitable server. They are meant to inform the receiver that it has been chosen to perform server tasks for the network and ask him to switch its role to 'server'. To account for packets to get lost during wireless transmission, tickets are resent by the server selection algorithm as long as the prospective server does not change its role to 'server'. In a static scenario like the two we are discussing right now, we would expect only a very small number of tickets sent by a node because their is no mobility and the network itself does not change. Nevertheless, we see in Figure 6.6 a larger number of ticket transmissions per node for scenarios with 36 or more nodes. In our implementation of the server selection algorithm, a server immediately changes its role if it receives a ticket packet from another node. If a ticket got lost during transmission or if the following announcement message by the prospective server does not reach our node, the ticket is resent automatically. Hence, it is logical to assume that the increase in tickets is caused by packet loss. In general, we notice that the communication overhead grows linearly as the number of nodes are increased quadratically. In contrast to our results shown here, it continues to increase in the same way with a growing number of nodes. But as more and more frames get lost due to network overload, we see a reduction in overhead. This is because in reality we added up all successfully received data of a node only.

In the rectangular scenario, our chosen interval between announcement messages is therefore not ideal as it causes an network congestion in case of a densely packed group of nodes in a small area with no movement. The announcement message interval determines how quickly a node can detect changes in the network. A small value means a faster reaction time of the server selection algorithm at the expense of higher overhead. This means that in scenarios with no or low mobility, the announcement interval can be set to a higher value. Still, we have decided to keep the interval at the same setting which we will use throughout our evaluation in order to be able to directly compare our results in different scenarios. Nevertheless, Figure 6.5 generally shows that even with this configuration, the total overhead caused by the server selection algorithm is low.

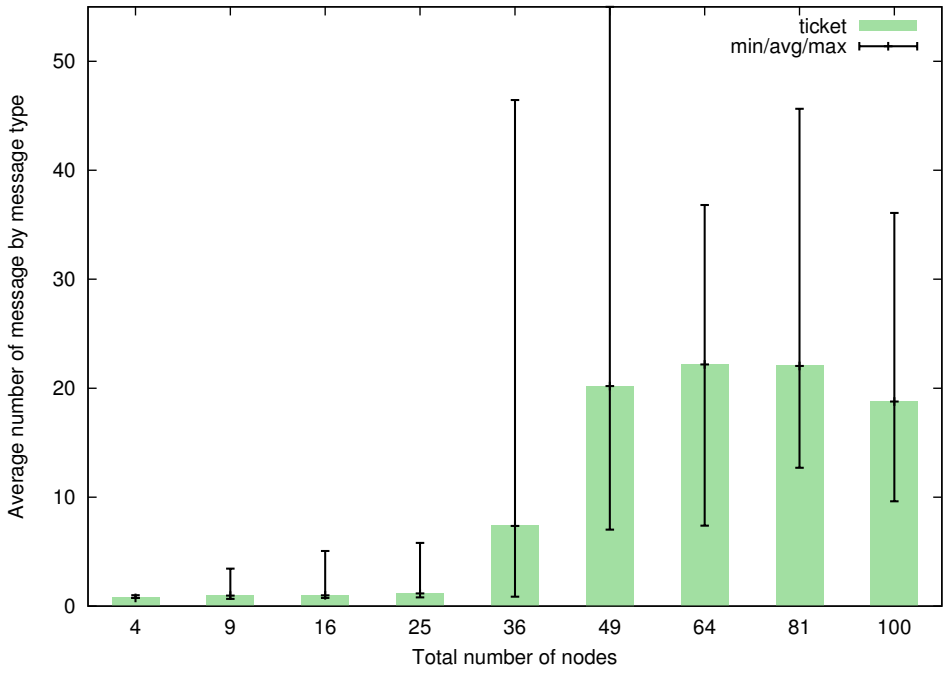
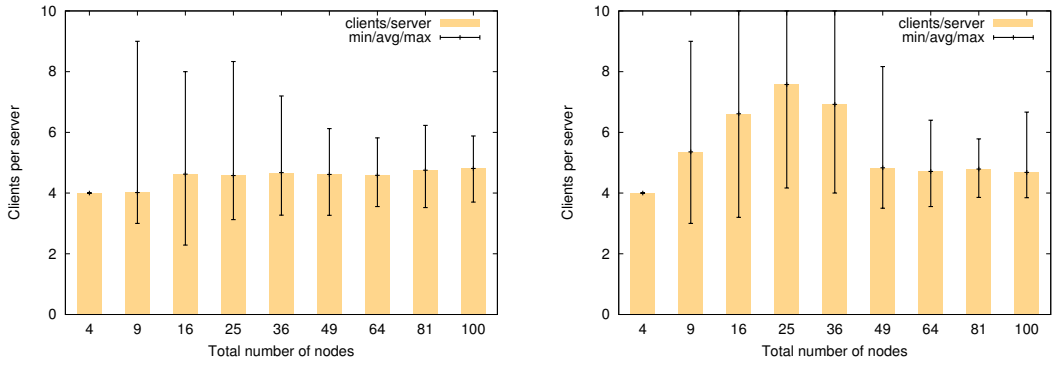


Figure 6.6.: Average number of tickets send per node (rectangular scenario)

For the linear scenario in Figure 6.5(a), the overhead increases as more nodes are introduced and later remains constant as the total number of nodes is increased further. This result is consistent with the average number of neighbours per node. In fact, the average overhead per node even decreases slightly with an increasing number of nodes. The reason for this is the hidden terminal problem which we encounter in this scenario. As nodes are arranged like pearls on a string in this scenario, the average distance between them is higher than in the rectangular scenario. Thus, it is not always possible for a node to reliably detect if the wireless medium is busy or not. Also, as we use broadcasts to send our announcement messages, RTS/CTS cannot be used to notify hidden terminals of transmissions of other nodes. However, this drop in overhead is barely noticeable and the algorithm scales very well in this scenario.

Finally, we take a look at the number of servers that are created by the algorithm. To allow for a better comparison between scenarios with a different number of nodes we decided to show the average number of clients per server rather than the total number of servers. Figure 6.7 shows these results again for the linear and for the rectangular scenario. For the linear scenario, we have a constant number of four to five clients per server. The two leftmost results are slightly smaller because the number of servers as well as the total number of clients is low. In the rectangular scenario, we notice a



(a) Nodes in a line

(b) Nodes in a rectangle

Figure 6.7.: Clients per Server

similar pattern as seen in Figure 6.5(b). What happens here is that when the number of nodes is low, a single server can provide services to all nodes in the network. As the number of nodes increase, more servers are chosen but still some of our 50 simulations can work with only a single server. This behaviour depends on how the random node weights are distributed in the individual simulation. Still, the more nodes are present in the network the less likely it is that only a single server is determined. As the network size and also the packet loss increases, the number of clients per server comes back down to a stable level that is similar to the results in the linear scenario.

From our analysis we conclude that the server selection algorithm scales well. It has a low communication overhead which for the most part is unaffected by an increasing number of nodes. In all scenarios the total traffic at any node is less than 1,500 bytes/s. Furthermore, it determines a suitable number of servers and the average number of clients per server is not dependent on the size of the network. We have also seen that additional factors like network geometry and load can have a noticeable impact on how many servers are created although the performance of the server selection algorithm still remains reasonable under unfavourable conditions.

6.2.2. The Schoolyard Scenario

The schoolyard scenario is our first scenario for realistic mobile gaming. It comprises an area of 200 m by 100 m and a number of stationary gaming nodes as well as mobile background nodes. The initial position of all nodes is chosen randomly for every simulation run and the background nodes are moved around according to the Gauss-Markov mobility algorithm which we use to model pedestrian-like movement. For

path loss and shadowing deviation settings for the signal propagation algorithm, we used the values from our WLAN stack comparison in Section 6.1.3 which describe a direct line-of-sight communication in an urban environment. Finally, no obstacles are placed in this scenario as NS-2 does not support this feature. The simulation time for the schoolyard scenario is 15 minutes. More details on this scenario can be found in Chapter 2.

Table 6.7 shows the simulation results of the schoolyard scenario. It shows the different numbers of game and background nodes. The background node traffic probability shows the probability that a background node will produce traffic to a random node in the network. The individual results for each parameter settings are divided into two lines. The upper line shows results from our server selection algorithm namely the server determination time which is the time it takes from algorithm startup until it determines a server and the average number of clients per server. The second line shows the simulation result from the game application's point of view. The first value is the playability time which is calculated as the average fraction of simulation time that a node is in a playable state[1]. A higher value means a better playability for the network in general and is a general metric to determine application quality although the experience of an individual node may differ. The last value is the fraction of simulation time that the node is in the lost state which means that the node has no solution from the server selection algorithm. Ideally, the lost time should be low in our evaluation as to indicate a continuous availability of a server. Each result represents the average value from 20 simulation runs with the same parameter set but different random seeds. We use similar tables to show the evaluation results for other scenarios and we have chosen to present the results in this form so that they can be directly compared across scenarios.

We now take a closer look at the results from the schoolyard scenario and explain the reasons behind them. We start by looking at the determination time of the server selection algorithm. We can see that the highest value is a couple of seconds which decreases as the total number of game nodes increases. The reason for that is that a certain node density and good network coverage is necessary for the server selection algorithm to find a fast solution for every node. For our simulation runs with few game nodes this is not always the case. Here, a single node that is far away from all other nodes has difficulties in detecting them due their distance in the mobile network. We have taken a closer look at the values for individual simulation runs and noticed that it takes tens of seconds for these remote nodes to discover their neighbours and then for the algorithm to calculate a solution while all other nodes have a determination time below half a second. And because we use an average value of server determination

[1]The node states 'playable' and 'lost' are described in more detail on page 150.

Game nodes	0 background nodes	5 background nodes	10 background nodes
	Background node traffic probability 0%		
5	0.4s / 3.7 **92.9%** / 2.0%	1.7s / 3.6 **88.9%** / 2.5%	5.3s / 3.5 **90.1%** / 2.7%
10	1.1s / 3.8 **89.7%** / 0.4%	1.1s / 3.9 **91.5%** / 0.1%	5.5s / 3.9 **88.7%** / 1.1%
15	0.6s / 2.9 **81.75%** / 0.3%	0.4s / 3.0 **84.6%** / 0.4%	0.4s / 3.5 **87.8%** / 0.3%
20	0.5s / 4.3 **86.4%** / $<0.1\%$	0.4s / 4.1 **90.3%** / $<0.1\%$	0.4s / 4.0 **80.8%** / $<0.1\%$
25	0.4s / 3.9 **75.9%** / $<0.1\%$	0.4s / 3.6 **80.2%** / $<0.1\%$	0.4s / 4.3 **79.6%** / $<0.1\%$
	Background node traffic probability 50%		
5	0.4s / 3.8 **92.5%** / 2.1%	0.6s / 3.7 **91.8%** / 1.7%	4.3s / 3.2 **88.8%** / 3.3%
10	1.1s / 3.9 **90.4%** / 0.1%	0.6s / 3.9 **91.7%** / 0.7%	1.7s / 3.6 **87.0%** / 2.2%
15	0.6s / 2.9 **81.8%** / 0.3%	0.4s / 3.0 **86.7%** / 0.3%	0.7s / 2.9 **89.8%** / 0.4%
20	0.5s / 4.3 **86.4%** / $<0.1\%$	0.4s / 3.6 **88.3%** / $<0.1\%$	0.4s / 3.8 **83.5%** / $<0.1\%$
25	0.4s / 3.9 **75.2%** / $<0.1\%$	0.4s / 3.7 **80.5%** / $<0.1\%$	0.4s / 3.5 **76.9%** / $<0.1\%$
	Background node traffic probability 100%		
5	1.3s / 3.8 **89.9%** / 2.9%	3.3s / 4.0 **91.0%** / 1.3%	3.4s / 3.4 **87.2%** / 4.2%
10	1.1s / 3.9 **90.4%** / 0.1%	1.1s / 3.6 **91.1%** / 0.7%	1.2s / 3.5 **89.7%** / 0.3%
15	0.6s / 2.9 **81.8%** / 0.3%	0.5s / 3.1 **87.7%** / 0.3%	0.5s / 3.1 **85.5%** / 0.3%
20	0.5s / 4.3 **86.4%** / $<0.1\%$	0.4s / 3.3 **87.5%** / $<0.1\%$	0.4s / 3.9 **84.4%** / $<0.1\%$
25	0.4s / 3.9 **75.4%** / $<0.1\%$	0.4s / 3.6 **79.9%** / $<0.1\%$	0.4s / 3.6 **73.6%** / $<0.1\%$

Legend

avg. server determination time	/	avg. no. of clients/server	
Playable time	/	No server time	

Table 6.7.: Evaluation results: The schoolyard scenario

time here, remote notes have a significant high impact on this value when the total number of game nodes is low. We also calculated the 80 percentile value of the server determination time which is below two seconds for all simulation runs. We can see that as the number of game nodes increases, the server determination time comes down to a more familiar value. The server determination time is not affected much beyond normal fluctuation by the presence of background nodes regardless whether they create background traffic or not.

The average number of clients per server mainly remains stable between three and four with a few outliers above and below. There is no apparent pattern between the different simulation runs. This value is smaller than what we have previously seen in our scalability simulations in Section 6.2.1 which is reasonable as this scenario is large and not so densely populated by nodes. Next we take a look at the time where nodes are in lost state. Here, we notice higher values when the number of game nodes is low which again is owed to the fact that a single node is positioned apart from other game nodes. Like the server determination time, the lost time is also reduced significantly with an increasing number of game nodes. In this scenario the simulation time was 15 minutes which means that a value of 0.1 % accounts for approximately one second of simulation time.

Finally, we take a look at the playability time in the schoolyard scenario which looks at network quality from the applications point of view. We would like to point out and discuss two issues from our evaluation results. Firstly, we see that the values start at 90 percent for five game nodes and then decreases down to 73 percent as more game nodes are introduced. Secondly, we notice that the presence of background nodes and background traffic has no preeminent influence on our results. There are even some results which are better when more background nodes are configured to sent background traffic. When looking through the detailed results from our simulations, we discovered that the schoolyard scenario is, with some exceptions, mainly a single large collision domain at least when it comes to interference. This means that the majority of nodes are able to talk directly with other nodes while all data traffic can interfere with transmissions of other nodes. As a result, the network is more and more congested as more nodes and more traffic is introduced. In Figure 3.2 we have shown our theoretical analysis that the wireless channel is occupied for half the time if 20 games nodes transmit their game traffic at the highest possible data rate. This node number decreases further if some nodes transmit at lower speeds. For contention-based networks like WLAN 50 % channel occupancy time is a lot and results in noticeable degradation in network performance which is what we can see in our results as the number of game nodes increase. This simulation result confirms therefore our theoretical analysis that starting with 15 nodes the load of the network reaches a point where its load has a noticeable

impact on game traffic performance.

Again we see little impact of our background traffic in results. The first and foremost explanation for the lack of impact is that we use the TCP protocol to create background traffic in our simulations which dynamically adapts its throughput as not to overload the network. We have decided on using TCP for background traffic because it is the most common protocol used in the Internet today. The second reason is that there is already a number of constant bit rate UDP traffic from other game nodes present in the scenario. In our simulations of the schoolyard scenario the average throughput for all background traffic ranges from 57 to 142 kBit/s with peaks of individual connections that reach an average of up to 922 kBit/s during the 15 minute simulation run. As all background nodes move during the evaluation, we notice that node movement has a large influence on background traffic throughput. Due to throughput fluctuations that are caused by node movement our results are not conclusive as to how exactly a low number of nodes effects background traffic with respect to network connectivity. However, we notice that for most of the simulation runs, average background throughput is reduced noticeably as more nodes are introduced. This behaviour is coherent with our previous conclusions of network congestion. Finally, we would like to point out the fact that we created all simulation runs individually which includes movement and initial node placement with different random seeds. This is the reason why Table 6.7 shows different results for simulations with zero background traffic probability.

In Figure 6.8 we show the evaluation details of three selected simulation runs with five, ten and 25 game nodes respectively. It starts on the left with the activation of all game nodes and shows 120 seconds of simulation time. Due to readability issues if we would show the full data set, we decided to show the initial two minutes of each simulation run only. To visualize the status of each node, we use the colour code which we introduced in Section 6.1.4. We have chosen these three simulation runs because they are quite typical for their specific set of evaluation parameters. Before we look into details, we would like to mention that no conclusion can be drawn from the node numbers to their actual position on the schoolyard meaning that for example Node #1 and Node #2 do not necessarily be close together on the schoolyard. Figure 6.8(a) shows results from a small scenario of five game nodes and five background nodes all of which make traffic of their own. Soon after the game nodes were started, the server selection algorithm selects two nodes as servers. The first two nodes each select one of these servers and their game traffic remains optimal for most of the time. Only Node #1 experiences a brief moment of increased latency to its server. For Node #5 the situation is completely different as this node experiences problems right from the start of the simulation. Like the first two nodes, its server selection algorithm receives the announcements from at least one of the game servers and chooses the best one.

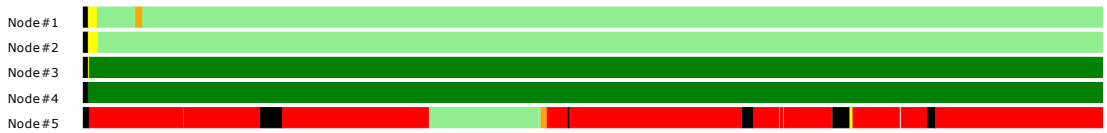
However, its game traffic experiences a loss rate of more than five percent for most of the time. The periods in black indicate that no announcement was received from an existing or prospective game server. Because announcements of the server selection algorithm are sent directly using broadcasts, we can rule out the possibility of routing problems as the reason behind this. It is more likely that Node #5 is positioned far away from the other game nodes and has only an intermittent connection to the rest of the network because the schoolyard area is only sparsely populated with the ten nodes in this simulation run. We believe that the brief period of optimal operation for Node #5 is caused by a passing background node.

We can draw several conclusions from this experiment. First of all and in certain situations server selection announcements can be received successfully even if network quality does not meet our game traffic requirements. We believe this is because these announcements are even shorter than the game traffic itself and by using broadcasts and consequently a low data rate, they are more resilient regarding outside interference. Hence, it is advisable for an interactive application that encounters bad quality for its traffic to switch to another server in the vicinity if available. Secondly, we believe that in our schoolyard scenario this problem can be addressed by the user of node #5 because he will move to a better spot if he experiences major network problems while other players encounter no problems at all.

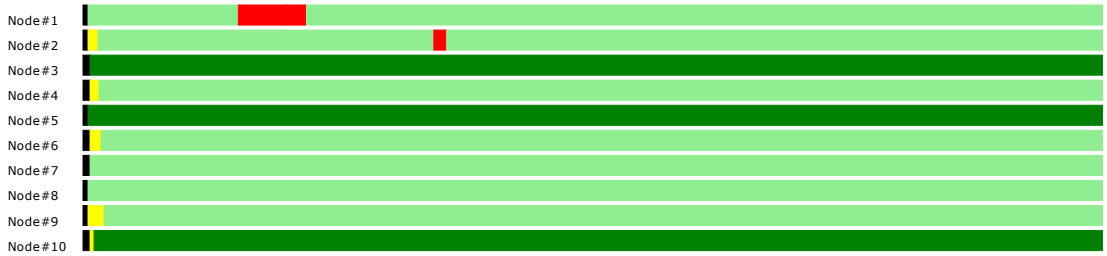
The scenarios with a medium number of nodes are presented in Figure 6.8(b). Here we can see that a suitable number of servers is chosen which remain stable throughout the shown period. We also notice that game traffic remains within the required boundaries and that only nodes #1 and #2 encounter some small time with high packet loss. In this simulation run, we have enough nodes to cover the schoolyard effectively so that game node density is sufficient to allow for a stable game for all nodes.

Finally, Figure 6.8(c) shows a schoolyard scenario simulation run with a high number of nodes. We first notice a lot of problems with high packet loss or high round-trip times throughout the picture. These problems are caused by congestion in the wireless network which is overloaded by traffic from that many nodes. We can also see that some clients experience only few or intermittent problems. We believe that those nodes are at the edge of the simulation area where interference is lower than in the middle and close to their respective zone servers so that the signal-to-noise ratio is comparatively high. Also, moving background nodes can influence different parts of the simulation area and their background traffic might help to push the game traffic above its limits.

We can also see in this figure that Node #8 was chosen as a game server and reverts back to client status. This status change is caused by the high loss which leads the server selection algorithm on Node #8 to believe that it has no clients anymore.



(a) 5 game nodes / 5 background nodes / 100 % background traffic probability



(b) 10 game nodes / 10 background nodes / 100 % background traffic probability

(c) 25 game nodes / 10 background nodes / 100 % background traffic probability

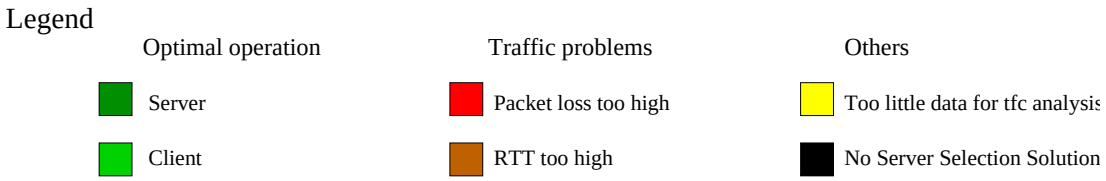


Figure 6.8.: Evaluation details of select scenarios (2 minute snapshot)

The next thing we look at is the time where the application is not usable because of network problems. We define a network outage for a node as the duration starting from the time when necessary application-specific quality-of-service limits (in our case RTT and packet loss) are exceeded until the time that they are met again. Figure 6.9 shows a cumulative distribution function of network outage durations for all 45 schoolyard simulations with different settings of node numbers and background traffic. We notice that more than 90 % of all outages are smaller than a quarter of a second and less than five percent take one second or longer. We can also identify the results with significant longer outages as the simulation runs with few nodes. Finally, we analysed the outages to see if we are able to find peaks or other anomalies that may hint at mobility, routing or other network problems. But we were only able to find that the percentage of outages below 250 ms increases with higher number of games nodes which again hints at network problems when a high number of game nodes is present.

In summary, the server selection algorithm and the application work quite well in the schoolyard scenario. We identified two problems regarding node connectivity and network congestion that show the limits of WLAN rather than our approach.

6.2.3. The Train Scenario

The second scenario we look at is the train scenario. Here we have a narrow but long simulation area of 3×175 meters which unlike the schoolyard scenario restricts communication along the longitudinal axis. Also, the simulation time is one hour which is four times longer than in the schoolyard scenario. Another difference is that now game nodes and background nodes both have fixed positions which represent their users sitting in their seat on the train. Due to a restriction of the network simulator NS-2, we could not model any wireless signal attenuation between coaches which have noticeable impact on signal strength in reality. Instead, we have increased the path loss to account for obstacles such as seats, people, and internal train structures in general. We have determined those through measurement in a regional train of Deutsche Bahn between Braunschweig and Hannover. Again, we show our results by using average values of 20 simulation runs per scenario setting.

Table 6.8 shows the evaluation results of the simulation of the train scenario. Although, we are using the same presentation of proportions to distinguish between playable and non-playable as in the schoolyard scenario, it should be noticed that one percent of simulation time in the train scenario is actually longer than in the schoolyard scenario because we now simulate a full hour instead of fifteen minutes. This means while the numbers can be compared directly between scenarios, the actual user experience may

differ. However, this is the case for all our results because we look at averages and not at the individual user.

The train scenario starts with the same familiar pattern that we already recognized in the schoolyard scenario, namely that it is difficult to achieve sufficient network connectivity with five game nodes only. Again, we see that in this case playability is low and the server determination time is high. For those areas where game nodes can reach each other, the server selection algorithm still selects a suitable number of servers with about 2.5 clients per server. Playability jumps up with ten game nodes as network connectivity between the nodes increases though some nodes still have connectivity problems as can be seen by the high server determination time and the larger than usual time that a node has no server. With ten game nodes we also notice by looking at the detailed results that some nodes had excellent connectivity while others where still struggling with network connectivity. Starting with 15 game nodes, we have sufficient node density to allow good network quality for all nodes which improves further with 20 game nodes. In contrast to the schoolyard scenario, we do not see any major congestion in the train scenario with 20 or 25 game nodes because traffic at one end of the train does not affect nodes at the other end. Thus, the train scenario is a multi-collision domain scenario and as such has a higher network capacity than the schoolyard allowing more game nodes to operate in a satisfactory manner. On the other side, the train scenario needs more game nodes in order to form a connected network and to be playable at all.

The server selection algorithm exhibits similar characteristics in both scenarios. If connectivity is low, the determination time can be very high up to the point where no server can be selected at all. The server determination time for the simulation runs with a higher number of game nodes is comparable to the schoolyard scenario with about half a second to determine or select a suitable server. Still, the number of clients per server is generally lower than in the schoolyard scenario which is owed to the fact that we adjusted the range of the wireless radios to simulate for obstacles as mentioned before and that the train scenario has a different shape so that nodes are more spread out. Still, the server selection algorithm is able to achieve about 2.5 to 3 clients per server. As with the schoolyard scenario, we again see no large impact of background traffic.

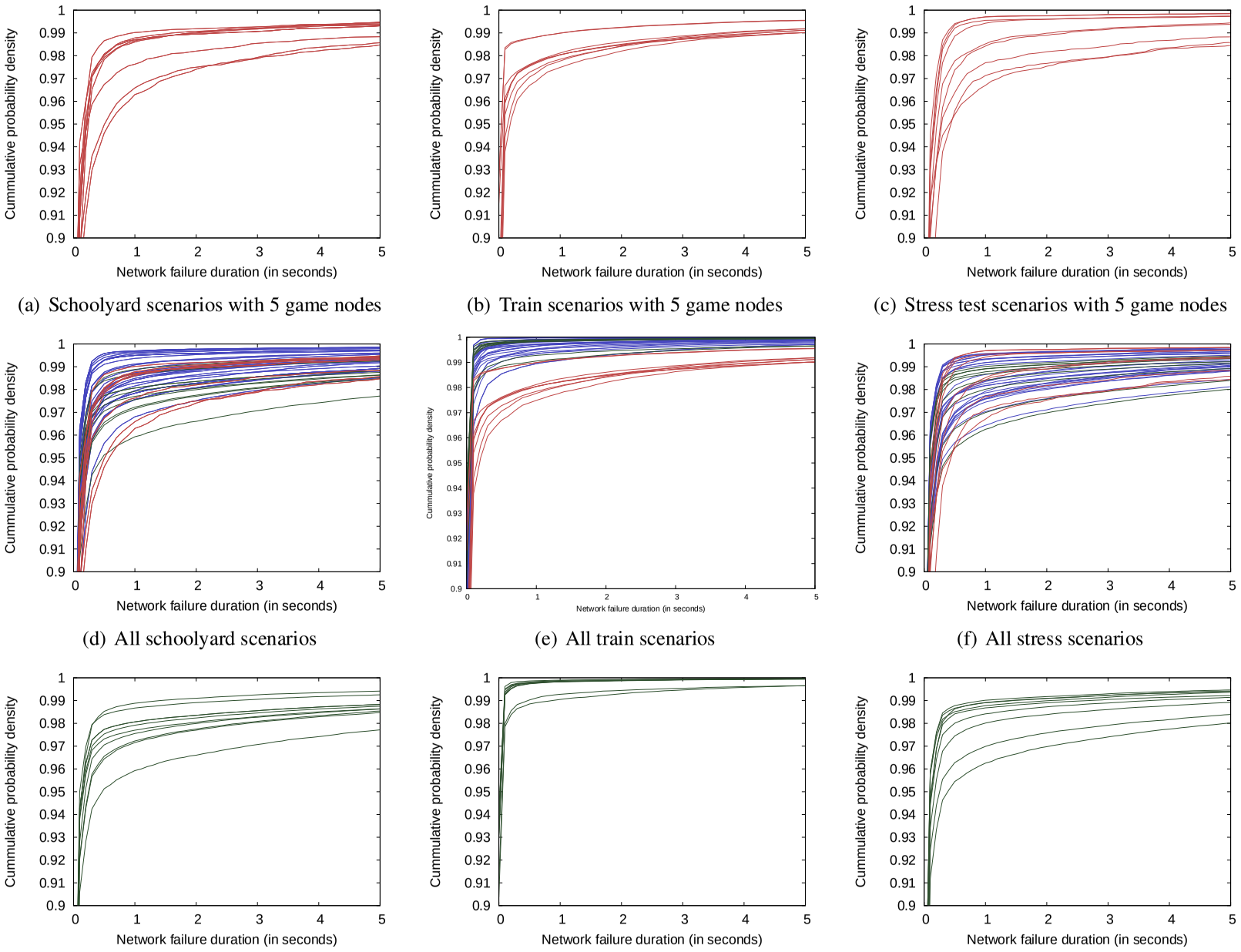
Figure 6.9(b) shows the CDF for network outages for the train scenario. We can see that the graph is steeper than in the schoolyard scenario which hints at the point that network congestion is of lesser importance here. Furthermore, we can clearly distinguish the simulation runs with a low number of game nodes where network connectivity problems have an impact on outages. The rest shows better performance than in the schoolyard scenario which is in line with our conclusions.

Game nodes	0 background nodes	5 background nodes	10 background nodes
	Background node traffic probability 0%		
5	79.3s / 2.7 **67.8%** / 14.4%	210.4s / 2.7 **66.7%** / 9.7%	215.1s / 2.6 **63.0%** / 17.8%
10	19.3s / 2.5 **91.7%** / 1.2%	6.3s / 2.3 **93.1%** / 0.4%	82.8s / 2.2 **87.9%** / 6.0%
15	1.0s / 2.9 **95.7%** / 0.3%	1.7s / 2.8 **94.2%** / 0.3%	1.6s / 2.5 **95.8%** / 0.9%
20	0.5s / 3.0 **98.4%** / <0.1%	0.4s / 2.9 **98.0%** / <0.1%	0.4s / 3.0 **98.0%** / <0.1%
25	0.42s / 3.2 **97.5%** / <0.1%	0.4s / 3.1 **97.4%** / <0.1%	0.5s / 2.9 **93.9%** / <0.1%
	Background node traffic probability 50%		
5	103.8s / 2.8 **67.4%** / 13.3%	138.1s / 2.4 **65.4%** / 15.5%	196.2s / 2.6 **61.9%** / 20.2%
10	16.2s / 2.5 **91.9%** / 1.0%	6.3s / 2.3 **93.1%** / 0.4%	82.8s / 2.2 **87.9%** / 6.0%
15	1.0s / 2.9 **95.7%** / 0.3%	1.8s / 2.8 **93.6%** / 0.9%	1.6s / 2.5 **96.3%** / 0.8%
20	0.4s / 3.0 **98.3%** / <0.1%	0.4s / 2.9 **98.1%** / <0.1%	0.4s / 3.0 **97.6%** / <0.1%
25	0.4s / 3.1 **97.6%** / <0.1%	0.4s / 3.0 **97.5%** / <0.1%	0.4s / 2.9 **98.5%** / <0.1%
	Background node traffic probability 100%		
5	108.6s / 2.7 **66.6%** / 14.2%	138.1s / 2.4 **65.4%** / 15.5%	124.3s / 2.2 **63.2%** / 23.2%
10	19.1s / 2.5 **92.4%** / 0.03%	7.2s / 2.3 **92.6%** / 0.3%	29.8s / 2.5 **91.9%** / 3.0%
15	1.1s / 2.9 **95.4%** / 0.4%	0.7s / 2.6 **95.1%** / 0.8%	1.8s / 2.4 **96.6%** / 0.8%
20	0.4s / 3.3 **98.2%** / <0.1%	0.5s / 2.9 **98.1%** / <0.1%	0.4s / 3.1 **97.5%** / < 0.1%
25	0.4s / 3.3 **97.6%** / <0.1%	0.4s / 3.0 **97.3%** / <0.1%	0.4s / 2.8 **96.4%** / 0.4%

Legend

avg. server determination time	/	avg. no. of clients/server
Playability time	/	No server time

Table 6.8.: Evaluation results: The train scenario



(a) Schoolyard scenarios with 5 game nodes

(b) Train scenarios with 5 game nodes

(c) Stress test scenarios with 5 game nodes

(d) All schoolyard scenarios

(e) All train scenarios

(f) All stress scenarios

(g) Schoolyard scenarios with 25 game nodes

(h) Train scenarios with 25 game nodes

(i) Stress test scenarios with 25 game nodes

Figure 6.9.: Cumulative probability density function of network outages

Summarising the train scenario, we have shown that we need at least ten to 15 game nodes in a common inter-city train to provide the coverage needed for a suitable network quality. Due to the fact that our network simulator does not support real obstacles, we expect this number to be rather optimistic. In Chapter 2 we assumed a seat occupancy ratio of $^2/_3$ for our train resulting in 287 people total. Assuming further that we need at least 20 people with game nodes to allow sufficient connectivity, we require a game node penetration ratio of at least 7 % of all people on board. Also and in contrast to the schoolyard scenario, people cannot move to a different position in the train as easy as before because they have seat reservations, heavy luggage or they cannot move to first class with a second class ticket. But apart from the connectivity problem with low number of game nodes, the train scenario performs surprisingly well because its reduced wireless transmission ranges reduce a node's range of interference. For the server selection algorithm, the train scenario is more challenging mainly because of it's one-dimensional shape. Still, it can provide satisfactory performance for individual nodes under problematic network conditions.

6.2.4. Stress Test Scenario

The stress test is the final scenario in our evaluation. In contrast to our initial belief that game nodes are static, we wanted to create a fully mobile scenario in which we can test the performance of our server selection algorithm. Thus, we again simulate the schoolyard scenario but with all game nodes moving around according to the Gauss-Markov algorithm. Table 6.10 shows the results of the stress test scenario. As expected the performance is generally similar to the schoolyard scenario. At five game nodes we start with good results which are a bit impaired by reduced network connectivity between the game nodes. Like in the schoolyard scenario, the average server determination time and the time without a server comes down as the number of game nodes increase. At the same time, playability decreases because of increased network load. Generally, we can say that game node mobility does not have a significant impact on the game playability or the performance of the server selection algorithm in this scenario. However, we see an increased variance in the simulation results with either decreased or increased performance between different simulation runs with the same parameter set.

We calculated the differences in playability time between the original schoolyard and the stress test scenario to see if we can determine a pattern for these differences. They are shown in Table 6.9 taking the schoolyard scenario as base. By comparing results from both scenarios, we noticed a slight performance advantage in the schoolyard scenario with increased background traffic. We haven taken a deeper look into the background traffic details to see whether it can explain these differences. Generally,

Game nodes	0 background nodes	5 background nodes	10 background nodes
	Background node traffic probability 0%		
5	-3.2%	-3.1%	-3.7%
10	+3.9%	-2.9%	+1.7%
15	+4.0%	+0.1%	-8.1%
20	-2.3%	-11.3%	+0.4%
25	+8.7%	-6.6%	-2.1%
	Background node traffic probability 50%		
5	-2.1%	+0.8%	+2.3%
10	+3.2%	-1.9%	+1.5%
15	+4.3%	-5.6%	-6.4%
20	-2.3%	-7.9%	+2.9%
25	-1.9%	-2.3%	-0.8%
	Background node traffic probability 100%		
5	+1.5%	-0.2%	-1.6%
10	+3.4%	-3.6%	-1.7%
15	+4.5%	-6.1%	+2.2%
20	-4.9%	-7.9%	-2.1%
25	+7.5%	-4.8%	+3.0%

Table 6.9.: Playability time differences between schoolyard and stress scenario

background traffic is highly variable in both scenarios due to the variable initial placement and the mobility of background nodes. However, the average background traffic throughput is about the same in both scenarios with only a few minor exceptions. Hence, there is no direct correlation in either direction between the variation of background traffic throughput and playability time. Furthermore, an analysis of the game traffic itself gave us no additional insight on what might have caused the fluctuation of our simulation results.

Concluding the simulation results from the stress test scenario, we see no noticeable impact on performance due to mobility of the game nodes. Especially playability, the average number of clients/server or the server determination time are well comparable with our results from the schoolyard scenario which are good for a medium number of nodes and at least sufficient for a low or high number of nodes. However, variation caused by node mobility may have an individual effect on the network performance perceived by the individual user so that his experience may be different.

6.3. Chapter Summary

In the evaluation, we started by looking at different methods to determine the performance of the server selection algorithm and a multi-player game application in an ad-hoc scenario. We found that simulation is a good trade-off of all evaluation methods because it offers reproducible results with good time accuracy which is important when measuring latency-sensitive applications. We then looked at two different NS-2 simulators with different WLAN stacks in detail and compared their results with a simple real world measurement to get an impression of their accuracy. We found that both simulators had various shortcomings regarding their available propagation model, variable changes in data rates between multiple transmissions, and how retransmissions are handled. We found that especially the latter two would be important for latency-sensitive applications as they have a direct influence on latency, latency variation and packet loss. By comparison we found that our improved version of NS-2 using the INRIA WLAN stack provided the most accurate replay of our measurement.

We then described our example application which was designed to resemble the network behaviour and traffic characteristics of the multi-player game 'CounterStrike' which is well-discussed in the literature. We also showed our evaluation method using node colouring and calculating average results for individual parameter settings to present our evaluation results so that they are easily comprehensible and comparable between different scenarios. We split our simulations in two parts. We started by looking at two artificial scenarios to determine the general overhead and scalability of

Game nodes	0 background nodes	5 background nodes	10 background nodes
	Background node traffic probability 0%		
5	1.7s / 3.4 **89.7%** / 1.8%	0.7s / 3.6 **92.0%** / 1.6%	0.7s / 3.8 **93.8%** / 2.5%
10	2.8s / 3.9 **93.6%** / 1.1%	0.5s / 2.9 **88.6%** / 1.1%	3.4s / 3.5 **90.4%** / 1.2%
15	0.4s / 3.6 **85.8%** / 0.2%	0.7s / 3.6 **84.7%** / 0.1%	0.5s / 3.7 **79.7%** / 0.4%
20	0.4s / 3.9 **84.1%** / 0.1%	0.4s / 3.7 **79.0%** / 0.1%	0.4s / 3.7 **81.2%** / 0.1%
25	0.4s / 4.2 **84.2%** / 0.1%	0.4s / 4.3 **73.6%** / 0.2%	0.4s / 4.4 **77.5%** / 0.1%
	Background node traffic probability 50%		
5	1.7s / 3.6 **90.4%** / 1.8%	0.8s / 3.8 **92.6%** / 1.7%	2.5s / 3.3 **91.1%** / 3.6%
10	3.1s / 4.0 **93.6%** / 1.2%	0.7s / 3.0 **89.8%** / 1.2%	1.3s / 3.7 **88.5%** / 0.6%
15	0.4s / 3.7 **86.1%** / 0.2%	0.4s / 3.4 **81.1%** / 0.1%	0.6s / 3.1 **83.4%** / 0.4%
20	0.4s / 3.9 **84.1%** / 0.1%	0.4s / 3.5 **80.4%** / 0.1%	0.4s / 3.2 **80.6%** / 0.4%
25	0.4s / 4.3 **83.3%** / 0.1%	0.4s / 4.0 **78.2%** / 0.2%	0.4s / 3.8 **76.1%** / 0.1%
	Background node traffic probability 100%		
5	1.7s / 3.7 **91.4%** / 1.8%	1.4s / 3.6 **90.8%** / 2.0%	3.3s / 3.1 **88.8%** / 2.6%
10	5.1s / 4.0 **93.8%** / 1.6%	2.7s / 3.3 **87.5%** / 1.4%	0.7s / 3.2 **88.0%** / 1.0%
15	0.4s / 3.6 **86.3%** / 0.2%	0.4s / 3.4 **81.6%** / 0.1%	0.6s / 3.1 **87.7%** / 0.4%
20	0.4s / 4.0 **81.5%** / 0.1%	0.4s / 3.3 **79.6%** / 0.2%	0.4s / 3.2 **82.3%** / 0.5%
25	0.4s / 4.2 **82.9%** / 0.1%	0.4s / 4.3 **75.1%** / 0.1%	0.4s / 3.5 **76.6%** / 0.2%

Legend

avg. server determination time	/	avg. no. of clients/server
Playability time	/	No server time

Table 6.10.: Evaluation results: The stress test scenario

the server selection algorithm before moving on to three real-world scenarios. In the artificial scenarios, we found that the server selection algorithm's performance is good in a single or multiple collision domain scenario as well as in a sparsely or densely populated area. In all situations, it was able to determine a suitable number of servers with the client/server ratio usually above four in less than a third of a second. Also, our server selection algorithm sends only very few data itself (less than 100 bytes/s per node) which remains nearly constant if the total number of network node increases. In all simulations the combined traffic (sending and receiving) at any node was always below 1.5 kBytes/s.

We then moved on evaluating our example application in a schoolyard and in a train scenario. Both scenarios included moving nodes that introduced TCP background traffic whereas the game nodes remained on their initial random position. We found the performance of the application as well as the server selection algorithm to be good often allowing network traffic between server and clients to match the QoS requirements of the application in more than 85 % of the simulation time. The remaining times were often small network problems that lasted less than a second. We also discovered some problematic situations where the network was too sparsely populated to maintain a good network connection among the game nodes. In this case, the server selection algorithm took a long time on average to find a server if it was found at all. Furthermore, we confirmed our justified assumption from Chapter 3 regarding network overload with more than 20 nodes. Our simulation data showed a noticeable degradation of game performance with 25 game nodes. Still, the server selection algorithm performed well in this overload situation. In the train scenario the performance for a high number of game nodes was better and showed no service degradation at all. We found the reason to be the presence of multiple collision domains in the train which in sum provide more network bandwidth. In this case, the smaller transmission ranges of the wireless radios were actually an advantage over the long-range transmission in the open-area schoolyard. Still, network connectivity problems for simulations with five game nodes existed in both scenarios.

Our final scenario was the so called stress scenario, an amended schoolyard scenario, in which the game nodes now moved as well. The results showed many similarities with the original schoolyard scenario, although the variance for the individual results where higher due to increased node mobility. In all scenarios, the server selection algorithm was able to find suitable servers that allowed at least sufficient playability. Without regarding the simulation runs with only a few game nodes that had connectivity problems, servers were determined in less than half a second and each server had between 2.5 and 3.5 clients on average.

7. Conclusion and Outlook

Wireless networks have become an important part of our life providing telephony and data services virtually everywhere. From an economical point of view it is, however, unreasonable to deploy a dense network of base stations nationwide to provide for blanket coverage high-speed data access everywhere. As a result, in rural areas or during travel it can be challenging to keep a constant high speed data connection to a fixed network infrastructure. From a technological point of view, direct communication between mobile devices that are in close proximity is a more efficient way of using the wireless medium instead of transmitting through intermediary base stations. Mobile ad-hoc networks (MANETs) allow this form of free and direct communication. Still, the question remains if they can achieve sufficient performance to support low-latency interactive applications.

In this thesis, we analysed the network requirements of latency-sensitive applications and the delay of wireless LANs. To achieve the necessary performance in mobile ad-hoc networks, we proposed several quality of service enhancements. By introducing our zone server approach, we create a hybrid multi-server architecture that is able to cope with high application requirements in a changing environment. With its server selection algorithm, a number of suitable, redundant zone servers are determined and maintained throughout the running period of the application. By using an enhanced simulator for mobile networks and realistic scenarios, we showed that interactive applications can be successfully used in MANETs.

7.1. Contribution

This thesis provides contributions in three main fields. The following list presents our achievements and provides a small summary for each contribution:

Latency and quality of service in mobile ad-hoc networks

- Analysis of latency in a mobile ad-hoc network

We modified a Linux WLAN driver to record detailed timing information in a real-time measurement. By correlating data from sender and receiver we were able to brake down end-to-end delay into basic elements and analyse the most prominent factors for latency in a WLAN ad-hoc network.

- Capacity estimation for a mobile gaming application

  By using a time budget model, we calculated the maximum number of players that a single WLAN collision domain can theoretically support. For the multi-player game 'CounterStrike' we found that IEEE 802.11b can support less than 16 players. The newer IEEE 802.11g can support a maximum of about 36 stations under ideal conditions.

- Quality of service enhancements for mobile ad-hoc networks

  Our preliminary evaluation showed that the network requirements of latency-sensitive applications cannot be met by a mobile ad-hoc network with background traffic. We introduced several QoS enhancements that achieve the necessary performance with a maximum of three hops. We also showed that AODV is the most suitable ad-hoc routing protocol and that only up to six low-layer retransmissions are beneficial for the application.

Server-based architecture for mobile ad-hoc networks

- The zone server architecture

  We introduced a novel architecture for mobile ad-hoc networks. The zone server approach combines client-server as well as peer-to-peer elements and supports mobility. Multiple zone servers are each responsible for distinct part of the network.

- The server selection algorithm

  The server selection algorithm is a fully distributed and scalable approach to select servers in the zone server architecture with low overhead. During node mobility the algorithm automatically adapts to changes in the network environment. We have build an implementation in C++ for the NS-2 simulator.

Simulation of mobile ad-hoc networks

- Specific scenarios for mobile gaming

  We defined the schoolyard during a break, a train station and a train ride as realistic scenarios for mobile gaming. We then use each scenarios to deduct

specific simulation parameters for our evaluation. For the train scenario we also present real-world measurements.

- Improvements to the NS-2 simulator

  We improved the NS-2 simulator because we noticed various shortcomings during our preliminary evaluation. We implemented the ARP request retries so that TCP background traffic is established reliably. We also reduced the transmission rates of broadcasts, implemented ad-hoc beacons and adapted reception threshold to met the characteristics of a current interface card.

- Comparison of results from real-world measurements and network simulation

  We compared different WLAN stacks and settings for the NS-2 simulator to achieve a realistic evaluation in terms of latency and packet loss. Together with our own improvements we were able to recreate a real-world measurement in the simulator with an error of 4.4 % for latency and an absolute error of 0.7 % for packet loss.

- Evaluation of a mobile gaming application in MANETs

  We evaluated the zone server architecture in three different scenarios. For the most part, our results show that the application's network requirements can be fulfilled for 85 – 98 % of the simulation time. We identified network overload and node connectivity as restrictive factors in our evaluation.

7.1.1. Evaluation Results

In the first part of our evaluation, we tested the scalability of the server selection algorithm in a theoretical linear and rectangular scenario. We showed that the average zone server determination time and the data sent by each node remains constant when the total number of nodes increases. Also the number of zone servers in relation to the total number of mobile nodes was nearly constant when the number of nodes was increased. We concluded that the server selection algorithm is very scalable and produces low overhead.

In the second part of our evaluation, we analysed the performance of a fast-paced multi-player game in three different scenarios. In the schoolyard scenario, the server selection algorithm as well as the application performed well. The main problem occurred when the total number of nodes reached 25 and the performance suffered from an increasing congestion of application traffic in the network. In contrast, the train ride scenario performed better in terms of node scalability because here communication range is lower than in the open space schoolyard. Hence, the application performed

well even with 25 mobile nodes. However, network connectivity was a problem in the train ride scenario when the total number of nodes was low. Finally, we looked again at the schoolyard scenario but with higher node mobility and found similar results as before.

In conclusion, our evaluation showed that our approach is scalable and able to support interactive latency-sensitive applications in mobile ad-hoc networks. As wireless technology advances further and offers more network resources, future networks will be able to support a higher number of concurrent operating mobile nodes. Network connectivity problems may also be alleviated with better wireless transmission methods, however, the integration of existing infrastructure nodes might be be more helpful in these situations.

7.2. Future Work & Outlook

There is a famous quote from Leonardo da Vinci which says „Art is never finished, only abandoned". And because da Vinci can be called a part-time scientist from today's point of view, the same holds true for any scientific research as well. Consequently, a number of ideas exist to enhance or extend our work further which we point out in this section.

In this thesis, we measured a single-hop wireless LAN network and simulated more complex multi-hop networks with a larger number of nodes. Because of the lack of proper low-level measurement and analysis tools for off-the-shelf WLAN hardware, we developed our own tools and modified WLAN driver. We can build on this work to improve the frame logging mechanism for high packet rates and extend our latency analyser to support the inspection of multi-hop traffic. Together, these extensions would allow to measure not only multi-hop scenarios but also the latency of traffic in a congested environment. Both results could then be used to refine the parameters of our network simulation and test our approach more realistically.

In MANETs, many applications and protocols require information about their immediate neighbourhood. Examples are service discovery protocols, routing protocols, and our server selection algorithm. Usually, the required information is sent periodically by every node in the network. For network performance reasons, these announcement messages or beacons are kept very small. But in IEE 802.11 they still require a complete transmission cycle including inter-frame spaces, headers and possibly a congestion period. Furthermore, as they need to be broadcasted to other nodes the standard does not allow these frames to be sent with high transmission rates. If nodes start sending out multiple beacons for various purposes, the wireless medium is used

quite inefficiently. A better way is to introduce a general announcement protocol for mobile ad-hoc networks which can combine announcements of one node in a single message thus reducing the overhead of multiple transmissions. Moreover, it may be prudent to put this service on the data link layer so that regular IEEE 802.11 broadcast messages about the service set ID (SSID) can be included as well.

To prevent network overload, a transmission power management scheme is useful when lots of nodes are present in a mobile ad-hoc network. It decreases the transmission range so that the receiver can safely pick up the data while at the same time the congestion area of a transmission for other nodes is reduced. While this approach can increase the throughput in a wireless network and allow more concurrent mobile nodes, it also introduces additional risks for latency-sensitive applications. By restricting the transmission power of a sender, the signal becomes weaker at the receiver and the probability that it cannot be received correctly gets higher. This may lead to a higher packet loss or higher latencies and increased delay variation in cases where packets need to be retransmitted. Also, the load on the network is higher if the number of retransmissions increases. On the other side, a lower signal strength means less interference for other nodes and improves their chance of transmitting or receiving packets simultaneously, thus improving the throughput of the mobile ad-hoc network in general. More research in this area is needed to determine a good and suitable trade-off to improve network performance in MANETs.

Finally, the server selection algorithm can be extended to give special attention to fixed infrastructure nodes. Currently, the zone server architecture assumes that all nodes are potentially mobile. While this approach also works with fixed nodes, it does not take into account that fixed nodes usually have an unlimited amount of energy and often a fair amount of processing power as well. Thus, they could serve as server-only nodes that can always maintain their role even if no clients are currently nearby. With an uplink to the Internet, fixed nodes can also serve as a gateway between instances of an application running in the mobile ad-hoc network and in the Internet.

Bibliography

[1] C. Demichelis and P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)," Telecomitalia Lab, Ericsson IPI, RFC 3363, Nov. 2002.

[2] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The Effects of Loss and Latency on User Performance in Unreal Tournament 2003," in *Proceedings of the 3rd Workshop on Network and System Support for Games*, Aug. 2004, pp. 144–151.

[3] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of Factors Affecting Players' Performance and Perception in Multiplayer Games," in *Proceedings of the 4th Workshop on Network and System Support for Games*, Hawthorne, USA, Oct. 2005.

[4] ITU, "One-way Transmission Time," International Telecommunication Union, Recommendation ITU-T G.114, Jan. 2003.

[5] C. Chafe, M. Gurevich, G. Leslie, and S. Tyan, "Effect of time delay on ensemble accuracy," in *Proceedings of the International Symposium on Musical Acoustics (ISMA'04)*, Mar. 2004.

[6] C. Systems, "Voice Over IP - Per Call Bandwidth Consumption," Document ID 7934 at http://www.cisco.com/en/US/tech/tk652/tk698/technologies_tech_note09186a0080094ae2.shtml.

[7] J. Davidson, J. Peters, M. Bhatia, S. Kalidindi, and S. Mukherjee, *Voice over IP Fundamentals*. Cisco Press, Jul. 2007.

[8] Johannes Faerber, "Network Game Traffic Modelling," in *Proceedings of the 1st Workshop on Network and System Support for Games*, Apr. 2002, pp. 53–57.

[9] T. T. G. Jr. and E. R. Mann, "Cathode-Ray Tube Amusement Device," United States Patent Office, Patent # 2455992, Dec. 1948.

[10] T. Henderson, "The effects of relative delay in networked games," PhD Dissertation, University College London, Department of Computer Science, Feb. 2003.

[11] G. Armitage, M. Claypool, and P. Branch, *Networking and Online Games*. John Wiley & Sons Ltd, Chichester, England, Jun. 2006.

[12] Entertainment Software Association (ESA), "Essential Facts about the Computer and Video Game Industry – 2006 Sales, Demographic and Usage Data."

[13] G. Armitage, "An experimental estimation of latency sensitivity in multiplayer Quake 3," in *Proceedings of the 11th IEEE International Conference on Networks (ICON'03)*, Sep. 2003, pp. 137–141.

[14] P. Quax, P. Monsieurs, W. Lamotte, D. D. Vleeschauwer, and N. Degrande, "Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game," in *Proc. Workshop on Network and System Support for Games*, 2004, pp. 152–156.

[15] J. Nichols and M. Claypool, "The Effects of Latency on Online Madden NFL Football," in *Proc. of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Jun. 2004.

[16] A. F. Wattimena, R. E. Kooij, J. M. van Vugt, and O. K. Ahmed, "Predicting the perceived quality of a first person shooter: the quake iv g-model," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (Netgames'06)*. New York, NY, USA: ACM Press, 2006, p. 42.

[17] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu, "The Effect of Latency on User Performance in Warcraft III," in *Proceedings of the 2nd Workshop on Network and System Support for Games*, May 2003, pp. 3–14.

[18] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *The Psychological Review*, vol. 63, pp. 81–97, 1956.

[19] K. J. Meier and J. Bohte, "Ode to Luther Gulick: Span of Control and Organizational Performance," *Administration & Society*, vol. 32, no. 2, pp. 115–137, May 2000. [Online]. Available: http://dx.doi.org/10.1177/00953990022019371

[20] L. Pantel and L. Wolf, "On the impact of delay on real-time multiplayer games," in *Proc. of the 12th international Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, New York, USA, 2002, pp. 23–29.

[21] T. Fritsch, H. Ritter, and J. Schiller, "The effect of latency and network limitations on mmorpgs: a field study of everquest2," in *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA: ACM, 2005, pp. 1–9.

[22] T. Yasui, Y. Ishibashi, and T. Ikedo, "Influences of network latency and packet loss on consistency in networked racing games," in *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA: ACM, 2005, pp. 1–8.

[23] P. Branch and G. Armitage, "Extrapolating server to client ip traffic from empirical measurements of first person shooter games," in *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA: ACM, 2006, p. 24.

[24] S. L. Kent, *The Ultimate History of Video Games: From Pong to Pokemon–the Story behind the Craze That Touched Our Lives and Changed the World*. Rocklin, CA, USA: Prima Communications, Inc., 2001.

[25] P. Latimer, "Atari lynx," *Retro Gamer*, pp. 24–31, Jul. 2005.

[26] Alcar, "Sony Playstation Portable (PSP) Picture," http://de.wikipedia.org/w/index.php?title=Datei:Psp1.png, accessed 2011-02-23.

[27] Kudo-kun, "Nintendo DS Picture," http://de.wikipedia.org/w/index.php?title=Datei:Nintendo_DS_Trans.png, accessed 2011-02-23.

[28] Nintendo, "Consolidated Financial Highlights – Consolidated Results for the Six Months Ended September 2007 and 2008," Oct. 2008.

[29] M. Claypool, "On the 802.11 Turbulence of Nintendo DS and Sony PSP Handheld Network Games," in *Proceedings of the 4th Workshop on Network and System Support for Games*, Hawthorne, USA, Oct. 2005.

[30] *IEEE Std 802.11b-1999: Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band*, Jun. 2004.

[31] A. Akkawi, S. Schaller, O. Wellnitz, and L. C. Wolf, "A mobile gaming platform for the IMS," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games (Netgames'04)*, 2004, pp. 77–84.

[32] ——, "Networked Mobile Gaming for 3G-Networks," in *Proceedings of 3rd International Conference on Entertainment Computing (ICEC '04)*, 2004, pp. 457–467.

[33] T. Dreher, "Pervasive Games: Interfaces, Strategies and Moves," Published at IASLonline: http://iasl.uni-muenchen.de/links/NAPGe.html – last accessed on 2008-12-08.

[34] ——, "Sammeltipp 2: Spiele im Stadtraum, Teil 1," Published at IASLonline: http://iasl.uni-muenchen.de/links/TippSammel2.html – last accessed on 2008-12-08.

[35] A. D. Cheok, F. S. Wan, K. H. Goh, X. Yang, W. Liu, and F. Farbiz, "Human pacman: A sensing-based mobile entertainment system with ubiquitous computing and tangible interaction," in *Proceedings of 2nd ACM SIGCOMM workshop on Network and system support for games (Netgames'03)*, 2003, pp. 106–117.

[36] A. D. Cheok, K. H. Goh, W. Liu, F. Farbiz, S. L. Teo, H. S. Teo, S. P. Lee, Y. Li, S. W. Fong, and X. Yang, "Human pacman: A mobile wide-area entertainment system based on physical, social, and ubiquitous computing," in *Advances in Computer Entertainment Technology*, 2004, pp. 360–361.

[37] K. Mitchell, D. McCaffery, G. Metaxas, J. Finney, S. Schmid, and A. Scott, "Six in the city: introducing real tournament - a mobile ipv6 based context-aware multiplayer game," in *Proceedings of 2nd ACM SIGCOMM workshop on Network and system support for games (Netgames'03)*. New York, NY, USA: ACM, 2003, pp. 91–100.

[38] A. D. Cheok, K. H. Goh, W. Liu, F. Farbiz, S. W. Fong, S. L. Teo, Y. Li, and X. Yang, "Human pacman: a mobile, wide-area entertainment system based on physical, social, and ubiquitous computing," *Personal Ubiquitous Comput.*, vol. 8, no. 2, pp. 71–81, 2004.

[39] Siemens AG, Transportation Systems, "The Multiple-Unit Train for the European High-Speed Network – ICE 3 for German Rail and Netherlands Railways," Order No. A19100-V800-B248-V2-X-7600, Erlangen, Germany.

[40] "Wi-Fi access on ICE trains and in DB Lounges," http://www.bahn.de/i/view/GBR/en/trains/overview/wi-fi-access.shtml, More detailed german version available at http://www.bahn.de/p/view/service/zug/railnet_ice_bahnhof.shtml, both accessed 2011-06-30.

[41] *IEEE Std 802.3-2008: Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, Dec. 2008.

[42] *ANSI/IEEE Std 802.5-1998E(R2003): Part 5: Token ring access method and Physical Layer specifications*, May 1998.

[43] *ANSI/IEEE Std 802.11-1999: Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications* , Aug. 1999.

[44] *IEEE Std 802.11e-2005: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements*, Nov. 2005.

[45] *IEEE Std 802.1d-2004: Media Access Control (MAC) Bridges*, Jun. 2004.

[46] J. H. Schiller, *Mobile communications*, 2nd ed. Addison-Wesley, 2003.

[47] M. S. Gast, *802.11 Wireless Networks - The Definitive Guide*, 2nd ed. O'Reilly Media, Apr. 2005.

[48] P. Mockapetris, "Domain Names – Concepts and Facilities," Information Sciences Institute, University of Southern California, USA, IETF Request for Comment (RfC) No. 1034, Status: Standard, Nov. 1987.

[49] ——, "Domain Names – Implementation and Specification," Information Sciences Institute, University of Southern California, USA, IETF Request for Comment (RfC) No. 1035, Status: Standard, Nov. 1987.

[50] R. T. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," University of California, Compaq, World Wide Web Consortium, Xeorox, Microsoft, IETF Request for Comment (RfC) No. 2616, Status: Draft Standard, Jun. 1999.

[51] T. C. A. for Internet Data Analysis (CAIDA), "Packet size distribution comparison between internet links in 1998 and 2008," Online at http://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml (last modified 2009-05-11, accessed 2009-09-16).

[52] W. John and S. Tafvelin, "Analysis of Internet Backbone Traffic and Header Anomalies observed," in *7th ACM SIGCOMM conference on Internet measurement (IMC'07)*. New York, NY, USA: ACM, 2007, pp. 111–116.

[53] W.-c. Feng, F. Chang, W.-c. Feng, and J. Walpole, "Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment (IMW'02)*. New York, NY, USA: ACM, 2002, pp. 151–156.

[54] T. Lang, P. Branch, and G. Armitage, "A Synthetic Traffic Model for Quake3," in *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology (ACE'04)*. New York, NY, USA: ACM Press, 2004, pp. 233–238.

[55] ITU, "Pulse Code Modulation (PCM) of Voice Frequencies," International Telecommunication Union, Recommendation ITU-T G.711, Jun. 1990.

[56] H. V. Balan, L. Eggert, S. Niccolini, and M. Brunner, "An Experimental Evaluation of Voice Quality over the Datagram Congestion Control Protocol," in *Proceedings of the 26th IEEE Internation Conference on Computer Communications (INFOCOM'07)*, Anchorage, USA, May 2007, pp. 2009–2017.

[57] ITU, "Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction (CS-ACELP)," International Telecommunication Union, Recommendation ITU-T G.729, Jan. 2007.

[58] S. Pilosof, R. Ramjee, D. Raz, Y. Shavitt, and P. Sinha, "Understanding TCP fairness over Wireless LAN," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications (INFOCOM'03)*, vol. 2, Apr. 2003, pp. 863–872.

[59] N. Blefari-Melazzi, A. Detti, I. Jabib, A. Ordine, and S. Salsano, "TCP Fairness Issues in IEEE 802.11 Networks: Problem Analysis and Solutions Based on Rate Control," *IEEE Transactions on Wireless Communications*, vol. 6, pp. 1346–1355, Apr. 2007.

[60] D. J. Leith, P. Clifford, D. Malone, and A. Ng, "TCP fairness in 802.11e WLANs," in *Proceedings of the International Conference on Wireless Networks, Communications an Mobile Computing*, Jun. 2005, pp. 649–654.

[61] M. Seyedzadegan, M. Othman, S. Subramaniam, and Z. Zukarnain, "The TCP fairness in WLAN: A Review," in *IEEE International Conference on Telecommunications and Malaysia International Conference on Communications (ICT-MICC'07)*, May 2007, pp. 644–648.

[62] S. W. Kim, B.-S. Kim, and Y. Fang, "Downlink and uplink resource allocation in IEEE 802.11 wireless LANs," *IEEE Transactions on Vehicular Technology*, vol. 54, pp. 320–327, Jan. 2005.

[63] M. Bottigliengo, C. E. Casetti, C.-F. Chiasserini, and M. Meo, "Smart traffic scheduling in 802.11 WLANs with access point," in *Proceedings of the IEEE Semiannual Vehicular Technology Conference (VTC'03-Fall)*, vol. 4, Oct. 2003, pp. 2227–2231.

[64] J. Ha and C.-H. Choi, "TCP Fairness for Uplink and Downlink Flows in WLANs," in *Proceedings of the 49th Annual IEEE Global Telecommunications Conference (GLOBECOM'06)*, Dec. 2006, pp. 1–5.

[65] D.-Y. Kim, E.-C. Park, and C.-H. Choi, "Distributed Access Time Control for Per-Station Fairness in Infrastructure WLANs," *IEEE Transactions on Communications*, vol. E89-B, pp. 2572–2579, Sep. 2006.

[66] T. Razafindralambo and F. Valois, "Performance Evaluation of Backoff Algorithms in 802.11 Ad-Hoc Networks," in *Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks (PE-WASUN'06)*. ACM, 2006, pp. 82–89.

[67] N. H. Vaidya, P. Bahl, and S. Gupta, "Distributed fair scheduling in a wireless lan," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom'00)*. New York, NY, USA: ACM, 2000, pp. 167–178.

[68] S. J. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," in *Proceedings of the 13th IEEE International Conference on Computer Communications (INFOCOM'94)*, vol. 2, Toronto, Canada, Jun. 1994, pp. 636–646.

[69] G. Berger-Sabbatel, A. Duda, O. Gaudouin, M. Heusse, and F. Rousseau, "Fairness and its impact on delay in 802.11 networks," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM'04)*, vol. 5, Dallas, USA, Nov. 29–Dec. 3, 2004, pp. 2967–2973.

[70] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance anomaly of 802.11b," in *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. (INFOCOM 2003)*, vol. 2, San Francisco, USA, Mar. 30–Apr. 3, 2003, pp. 836–843. [Online]. Available: http://www.comsoc.org/confs/ieee-infocom/2003/papers/21_01.PDF

[71] C. Casetti and C.-F. Chiasserini, "Improving Fairness and Throughput for Voice Traffic in 802.11e EDCA," in *Proceedings of the 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'04)*, Sep. 2004, pp. 525–530.

[72] A. Kamerman and L. Monteban, "WaveLAN-II: A high-performance wireless LAN for the unlicensed band," *Bell Labs Technical Journal*, vol. 2, no. 3, pp. 118–133, 1997.

[73] M. Lacage, M. H. Manshaei, and T. Turletti, "IEEE 802.11 Rate Adaptation: A Practical Approach," in *7th ACM International Symposium on Modeling, analysis and Simulation of Wireless and Mobile Systems (MSWiM'04)*. New York, NY, USA: ACM, 2004, pp. 126–134.

[74] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The Impact of Multihop Wireless Channel on TCP Throughput and Loss," in *Proceedings of the International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03)*, 2003.

[75] G. Xylomenos and G. C. Polyzos, "TCP and UDP performance over a wireless LAN," in *Proceedings of the International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, Mar. 1999, pp. 439–446.

[76] H. Hlavacs, K. A. Hummel, A. Albl, S. Frandl, J. Kirchler, and B. Ilic, "Effects of WLAN QoS Degradation on Streamed MPEG4 Video Quality," in *Wireless Systems and Mobility in Next Generation Internet*. Heidelberg, Germany: Springer, Jun. 2004, pp. 152–165.

[77] L. Carvalho, J. Angeja, and A. Navarro, "A New Packet Loss Model of the IEEE 802.11g Wireless Network for Multimedia Communications," *IEEE Transactions on Consumer Electronics*, vol. 51, no. 3, pp. 809–814, Aug. 2005.

[78] G. Bianchi, F. Formisano, and D. Giustiniano, "802.11b/g Link Level Measurements for an Outdoor Wireless Campus Network," in *Proceedings of the*

2006 International Symposium on a World of Wirelss, Mobile and Multimedia Networks (WoWMoM'06). Washington, DC, USA: IEEE Computer Society, 2006, pp. 525–530.

[79] V. Lenders, J. Wagner, and M. May, "Measurements from an 802.11b Mobile Ad Hoc Network," in *Proceedings of th 2006 International Symposium on a World of Wirelss, Mobile and Multimedia Networks (WoWMoM'06)*. Washington, DC, USA: IEEE Computer Society, 2006.

[80] M. Li, F. Li, M. Claypool, and R. Kinicki, "Weather forecasting: predicting performance for streaming video over wireless lans," in *Proceedings of the international workshop on Network and operating systems support for digital audio and video (NOSSDAV'05)*. New York, NY, USA: ACM Press, 2005, pp. 33–38.

[81] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," *SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 121–132, 2004.

[82] T. Issariyakul, E. Hossain, and A. S. Alfa, "End-to-End Batch Transmission in a Multihop and Multirate Wireless Network: Latency, Reliability, and Throughput Analysis," *IEEE Transactions on Mobile Computing*, vol. 5, no. 9, pp. 1143–1155, Sep. 2006.

[83] Y. C. Tay and K. C. Chua, "A capacity analysis for the ieee 802.11 mac protocol," *Wireless Networks*, vol. 7, no. 2, pp. 159–171, 2001.

[84] D. Djenouri and N. Badache, "MANET: Selfish Behavior on Packet Forwarding," in *Encyclopedia of Wireless and Mobile Communications*, B. Furht, Ed. Auerbach Publications, Apr. 2008.

[85] M. Mauve, J. Widmer, and H. Hartenstein, "A Survey on Position-Based Routing in Mobile Ad-Hoc Networks," *IEEE Network Magazine*, vol. 15, no. 6, pp. 30–39, Nov. 2001.

[86] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward, "A Distance Routing Effect Algorithm for Mobility (DREAM)," in *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking (MobiCom'08)*, Oct. 1998, pp. 76–84.

[87] Y.-B. Ko and N. H. Vaidya, "Location-Aided Routing (LAR) in Mobile Ad Hoc Networks," *Wireless Networks*, vol. 6, pp. 307–321, 2000.

[88] J. Moy, "OSPF Version 2," Ascend Communications, Inc., IETF Request for Comment (RfC) No. 2328, Status: Standard, Apr. 1998.

[89] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," Juniper Networks, NextHop Technologies, Inc., IETF Request for Comment (RfC) No. 4271, Status: Draft Standard, Jan. 2006.

[90] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," *ACM SIGCOMM Computer Communication Review*, vol. 24, pp. 234–244, Oct. 1994.

[91] P. Jacquet, P. Muhlethaler, T. H. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized Link State Routing Protocol for Ad Hoc Networks," in *Proceedings of the IEEE International Multi Topic Conference on Technology for the 21st Century (INMIC'01)*, Dec. 2001.

[92] C. Adjih, T. H. Clausen, P. Jacquet, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot, "Optimized Link State Routing," INRIA, Center for Advanced Research in Engineering Pvt. Ltd., IETF Request for Comment (RfC) No. 3626, Status: Experimental, Oct. 2003.

[93] C. E. Perkins and E. M. Royer, "Ad-hoc On-Demand Distance Vector Routing," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (HotMobile'99)*, Feb. 1999, pp. 90–100.

[94] C. E. Perkins, E. M. Belding-Royer, and S. R. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," Nokia Research Center, University of California, University of Cincinnati, IETF Request for Comment (RfC) No. 3561, Status: Experimental, Jul. 2003.

[95] D. B. Johnson and D. A. Maltz, *Mobile Computing*. Kluwer Academic Publishers, 1996, ch. Dynamic Source Routing in Ad-Hoc Wireless Networks, pp. 153–183.

[96] D. B. Johnson, D. A. Maltz, and Y.-C. Hu, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," Rice University, Microsoft Research, University of Illinois, IETF Request for Comment (RfC) No. 4728, Status: Experimental, Feb. 2007.

[97] Z. J. Haas, S. Member, and M. R. Pearlman, "The Performance of Query Control Schemes for the Zone Routing Protocol," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 427 – 438, Aug. 2001.

[98] P. Sinha, R. Sivakumar, and V. Bharghavan, "CEDAR: a core-extraction distributed ad hoc routing algorithm," in *Proc. IEEE INFOCOM 1999*, Mar. 1999, pp. 202–209.

[99] J. Wu, "Dominating-set-based routing in ad hoc wireless networks," in *Handbook of wireless networks and mobile computing*. New York, NY, USA: John Wiley & Sons, Inc., 2002, pp. 425–450.

[100] E. M. Royer and C.-K. Toh, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks," *IEEE Personal Communication*, vol. 6, pp. 46–55, Apr. 1999.

[101] R. Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications," in *Proceedings of the 1st IEEE International Conference on Peer-to-Peer Computing (P2P'01)*, Los Alamitos, CA, USA, 2001.

[102] R. Steinmetz and K. Wehrle, "Peer-to-Peer-Networking & -Computing," *Informatik Spektrum*, vol. 27, no. 1, pp. 51–54, 2004.

[103] A. Oram, *Peer-to-Peer – Harnessing the Power of Disruptive Technologies*. O'Reilly Media, Feb. 2001.

[104] M. P. Singh, "Peering at Peer-to-Peer Computing," *IEEE Internet Computing*, vol. 5, no. 1, pp. 4–5, Feb. 2001.

[105] J. B. Postel and J. Reynolds, "File Transfer Protocol (FTP)," Information Science Institute, IETF Request for Comment (RfC) No. 1985, Status: Standard, Oct. 1985.

[106] J. G. Myers and M. T. Rose, "Post Office Protocol - Version 3," Carnegie-Mellon University, Dover Beach Consulting, Inc., IETF Request for Comment (RfC) No. 1939, Status: Standard, May 1996.

[107] M. R. Crispin, "Internet Message Access Protocol - Version 4rev1," University of Washington, IETF Request for Comment (RfC) No. 3501, Status: Proposed Standard, Mar. 2003.

[108] T. Ylonen and C. L. (editor), "The Secure Shell (SSH) Protocol Architecture," SSH Communications Security Corp., IETF Request for Comment (RfC) No. 4251, Status: Proposed Standard, Jan. 2006.

[109] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," in *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[110] L. Gong, "JXTA: A Network Programming Environment," *IEEE Internet Computing*, vol. 5, no. 3, pp. 88–95, 2001.

[111] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the ACM conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM'01)*, 2001, pp. 149–160.

[112] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'01)*. Springer Verlag, 2001, pp. 329–350.

[113] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A Resilient Global-scale Overlay for Service Deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 41–53, 2004.

[114] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01)*, San Diego, USA, 2001, pp. 161–172.

[115] M. Hauswirth and S. Dustdar, "Peer-to-Peer: Grundlagen und Architektur," *Datenbank-Spektrum*, vol. 13, pp. 5–13, May 2005.

[116] T. Repantis and V. Kalogeraki, "Data dissemination in mobile peer-to-peer networks," in *Proceedings of the 6th international conference on Mobile data management (MDM'05)*, May 2005, pp. 211–219.

[117] S. K. Goel, M. Singh, D. Xu, and B. Li, "Efficient Peer-to-Peer Data Dissemination in Mobile Ad-Hoc Networks," in *Proceedings of the International Conference on Parallel Processing*, 2002.

[118] W. Kellerer and R. Schollmeier, "Proactive search routing for mobile peer-to-peer networks: Zone-based p2p," in *Proceedings of the 5th Workshop on Applications and Services in Wireless Networks (ASWN'05)*, Jun. 2005.

[119] A. Klemm, C. Lindemann, and O. P. Waldhorst, "A special-purpose peer-to-peer file sharing system for mobile ad hoc networks," in *Proceedings of the IEEE Semiannual Vehicular Technology Conference (VTC'03-Fall)*, vol. 4, Oct. 2003, pp. 2758–2763.

[120] R. Schollmeier, I. Gruber, and F. Niethammer, "Protocol for peer-to-peer networking in mobile environments," in *Proceedings of the 12th International Conference on Computer Communications and Networks (ICCCN'03)*, Oct. 2003, pp. 121–127.

[121] H. Pucha, S. M. Das, and Y. C. Hu, "Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks," in *Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'04)*, Dec. 2004.

[122] ——, "Ekta+: Opportunistic Multiplexing in a Wireless DHT," in *Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking (MobiShare'06)*, 2006, pp. 69–71.

[123] T. Fuhrmann, "Performance of scalable source routing in hybrid manets," in *Proceedings of the Fourth Annual Conference on Wireless On demand Network Systems and Services*, Obergurgl, Austria, Jan. 24–26 2007, pp. 122–129.

[124] C. Cramer and T. Fuhrmann, "ISPRP: A Message-Efficient Protocol for Initializing Structured P2P Networks," in *Proceedings of the 24th IEEE International Performance, Computing, and Communications Conference (IPCCC'05)*, Apr. 2005.

[125] M. Gerla, C. Lindemann, and A. Rowstron, "P2P MANET's - New Research Issues," in *Perspectives Workshop: Peer-to-Peer Mobile Ad Hoc Networks - New Research Issues*, ser. Dagstuhl Seminar Proceedings, M. Gerla, C. Lindemann, and A. Rowstron, Eds., no. 05152. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[126] M. Mauve, "Consistency in replicated continuous interactive media," in *Proceedings of the 8th ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, 2000, pp. 181–190.

[127] L. Pantel and L. C. Wolf, "On The Suitability of Dead Reckoning Schemes for Games," in *Proceedings of the 1st ACM Workshop on Network and System Support for Games (Netgames'02)*, 2002, pp. 79–84.

[128] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in Dead-Reckoning Based Distributed Multi-Player Games," in *Proceedings of the 3rd ACM Workshop on Network and System Support for Games (Netgames'04)*, 2004, pp. 161–165.

[129] M. Mauve, S. Fischer, and J. Widmer, "A Generic Proxy System for Networked Computer Games," in *Proceedings of the 1st ACM Workshop on Network and System Support for Games (Netgames'02)*, 2002, pp. 25–28.

[130] E. Cronin, B. Filstrup, A. R. Kurc, and S. Jamin, "An Efficient Synchronization Mechanism for Mirrored Game Architectures," in *Proceedings of the 1st workshop on Network and System Support for Games (Netgames'02)*, 2002, pp. 67–73.

[131] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin, "An Efficient Synchronization Mechanism for Mirrored Game Architectures," *Multimedia Tools and Applications*, vol. 23, no. 1, pp. 7–30, 2004.

[132] S. D. Webb, S. Soh, and W. Lau, "Enhanced Mirrored Servers for Network Games," in *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games (Netgames'07)*, 2007, pp. 117–122.

[133] L. Gautier, C. Diot, and J. Kurose, "End-to-end Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet," in *Proceedings of the International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, 1999, pp. 1470–1479.

[134] L. Gautier and C. Diot, "Design and evaluation of mimaze, a multi-player game on the internet," in *International Conference on Multimedia Computing and Systems*, 1998, pp. 233–236.

[135] C. Diot and L. Gautier, "A Distributed Architecture for Multiplayer Interactive Applications on the Internet," *IEEE Networks magazine*, vol. 13, no. 4, pp. 6–15, Jul. 1999.

[136] J. Jardine and D. Zappala, "A Hybrid Architecture for Massively Multiplayer Online Games," in *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games (Netgames'08)*, 2008, pp. 60–65.

[137] H.-H. Lee and C.-H. Sun, "Load-balancing for peer-to-peer networked virtual environment," in *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games (Netgames'06)*, 2006, p. 14.

[138] C. GauthierDickey, V. Lo, and D. Zappala, "Using N-Trees for Scalable Event Ordering in Peer-to-Peer Games," in *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'05)*, 2005, pp. 87–92.

[139] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 353–366, 2004.

[140] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: A Distributed Architecture for Online Multiplayer Games," in *Proceedings of the 3rd Symposium on Networked Systems Design & Implementation (NSDI'06)*, May 2006, pp. 12–12.

[141] T. Iimura, H. Hazeyama, and Y. Kadobayashi, "Zoned Federation of Game Servers: a Peer-to-peer Approach to Scalable Multi-player Online Games," in *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games (Netgames'04)*, 2004, pp. 116–120.

[142] S. M. Riera, O. Wellnitz, and L. Wolf, "A zone-based gaming architecture for ad-hoc networks," in *Proceedings of the Workshop on Network and System Support for Games (NetGames2003)*, Redwood City, USA, May 2003.

[143] P. Kabus, W. W. Terpstra, M. Cilia, and A. P. Buchmann, "Addressing cheating in distributed mmogs," in *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games (Netgames'05)*, 2005, pp. 1–6.

[144] P. Kabus and A. P. Buchmann, "Design of a cheat-resistant p2p online gaming system," in *Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts (DIMEA'07)*, 2007, pp. 113–120.

[145] S. D. Webb and S. Soh, "Cheating in networked computer games: A review," in *Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts (DIMEA'07)*, 2007, pp. 105–112.

[146] N. E. Baughman and B. N. Levine, "Cheat-Proof Playout for Centralized and Distributed Online Games," in *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01)*, Apr. 2001, pp. 104–113.

[147] O. Wellnitz and L. Wolf, "Assigning Game Server Roles in Mobile Ad-hoc Networks," in *Proceedings of the 16th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'06)*, 2006, pp. 1–6.

[148] K. Farkas, T. Hossmann, B. Plattner, and L. Ruf, "NWC: Node Weight Computation in MANETs," in *Proceedings of the 16th International Conference on Communications and Networks (ICCCN'07)*, Aug. 2007, pp. 1059–1064.

[149] D. Budke, K. Farkas, O. Wellnitz, B. Plattner, and L. Wolf, "Real-Time Multiplayer Game Support Using QoS Mechanisms in Mobile Ad Hoc Networks," in *Proceedings of the 3rd Annual Conference on Wireless On demand Network Systems and Services (WONS 2006)*, Les Ménuires, France, Jan. 2006.

[150] S. Helal, N. Desai, V. Verma, and C. Lee, "Konark – A Service Discovery and Delivery Protocol for Ad-hoc Networks," in *Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC)*, New Orleans, Mar. 2003.

[151] E. Gutman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," Sun Microsystems, @Home Network, Vinca Corporation, IETF Request for Comment (RfC) No. 2608, Status: Proposed Standard, Jun. 1999.

[152] "The network simulator - ns-2," http://www.isi.edu/nsnam/ns/, accessed 2008-09-24.

[153] M. Claypool, D. LaPoint, and J. Winslow, "Network Analysis of Counter-strike and Starcraft," in *Proceedings of the 22nd IEEE International Performance, Computing, and Communications Conference (IPCCC'03)*, Apr. 2003, pp. 261–268.

[154] F. Kuhn and R. Wattenhofer, "Constant-time distributed dominating set approximation," in *Proceedings of the 22nd ACM International Symposium on the Principles of Distributed Computing*, Boston, USA, Jul. 2003.

[155] K. Farkas, "Supporting distributed services in mobile ad hoc networks," Dissertation, Swiss Federal Insitute of Technology, Zurich, Department of Computer Science, Nov. 2006.

[156] "The Multiband Atheros Driver for Wireless Fidelity (MadWifi)," http://madwifi.org.

[157] G. Armitage and L. Stewart, "Limitations of using real-world, public servers to estimate jitter tolerance of first person shooter games," in *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology (ACE'04)*, 2004, pp. 257–262.

[158] E. Nordström, "Implementation of the Ad-hoc On-demand Distance Vector routing protocol for Linux and NS-2," http://core.it.uu.se/core/index.php/AODV-UU, accessed 2010-03-23.

[159] F. J. Ros, "Implementation of the Optimized Link State Routing protocol for Linux and NS-2," http://masimum.dif.um.es/um-olsr/html/, accessed 2010-03-23.

[160] D. Budke, "Quality of Service for Multiplayer Provisioning in Mobile Ad Hoc Networks," Master's Thesis, Technische Universität Carolo-Wilhemina zu Braunschweig and Eidgenössische Technische Hochschule Zürich, Sep. 2005.

[161] K. Kowalik, B. Keegan, and M. Davis, "Making OLSR Aware of Resources," in *Proceedings of the International Conference on Wireless Communications (WiCom'07)*, Sep. 2007, pp. 1488–1493.

[162] Y. Xue, H. Jiang, and H. Hu, "Optimization on OLSR Protocol for Lower Routing Overhead," in *Proceedings of the Third International Conference on Rough Sets and Knowledge Technology (RSKT'08)*, May 2008, pp. 723–730.

[163] T. You, C.-H. Yeh, and H. Hassanein, "DRCE: a high throughput QoS MAC protocol for wireless ad hoc networks," in *Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC'05)*, Jun. 2005, pp. 671–676.

[164] C. R. Lin and M. Gerla, "Asynchronous multimedia multihop wireless networks," in *Proceedings of the 16th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'97)*, 1997, p. 118.

[165] P. Becker, R. Gotzhein, and T. Kuhn, "Macz - a quality-of-service mac layer for ad-hoc networks," in *Proceedings of the 7th International Conference on Hybrid Intelligent Systems (HIS'07)*, 2007, pp. 277–282.

[166] Q. Xue and A. Ganz, "Ad hoc qos on-demand routing (aqor) in mobile ad hoc networks," *Journal of Parallel and Distributed Computing*, vol. 63, no. 2, pp. 154 – 165, 2003.

[167] Y. Sakurai and J. Katto, "AODV multipath extension using source route lists with optimized route establishment," in *International Workshop on Wireless Ad-Hoc Networks (IWWAN'04)*, Jun. 2004, pp. 63–67.

[168] P. Yang and B. Huang, "Multi-path routing protocol for mobile ad hoc network," in *International Conference on Computer Science and Software Engineering*, vol. 4, 2008, pp. 1024–1027.

[169] H. Xiao, W. K. Seah, A. Lo, and K. C. Chua, "A flexible quality of service model for mobile ad-hoc networks," in *Proceedings of 51st IEEE Conference on Vehicular Technology (VTC-Spring'00)*, May 2000, pp. 445–449.

[170] J. Wroclawski, "The Use of RSVP with IETF Integrated Services," MIT, IETF Request for Comment (RfC) No. 2210, Status: Proposed Standard, Dec. 1998.

[171] K. Nichols, S. Blake, F. Baker, and D. L. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," Cisco Systems, Torrent Networking Technologies, EMC Corporation, IETF Request for Comment (RfC) No. 2474, Status: Proposed Standard, Dec. 1998.

[172] S. Blake, D. L. Black, M. A. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Service," Torrent Networking Technologies, EMC Corporation, Sun Microsystems, Nortel UK, Lucent Technologies, IETF Request for Comment (RfC) No. 2475, Status: Informational, Dec. 1998.

[173] X. Zhang, S.-B. Lee, G.-S. Ahn, and A. T. Campbell, "Insignia: An ip-based quality of service framework for mobile ad hoc networks," *Parallel and Distributed Computing, Special Issue on Wireless and Mobile Computing and Communications*, vol. 60, no. 4, pp. 374–406, Apr. 2000.

[174] Y. He and H. Abdel-Wahab, "Hqmm: A hybrid qos model for mobile ad-hoc networks," in *Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06)*, Apr. 2006, pp. 194–200.

[175] Gahng-Seop Ahn and Andrew T. Campbell and Andras Veres and Li-Hsiang Sun, "SWAN: Service Differentiation in Stateless Wireless Ad Hoc Networks," in *Proc. IEEE INFOCOM 2002*, Jun. 2002, pp. 457–466.

[176] K. Chen, S. H. Shah, and K. Nahrstedt, "Cross-layer design for data accessibility in mobile ad hoc networks," *Wireless Personal Communications*, vol. 21, no. 1, pp. 49–76, 2002.

[177] H. Lundgren, E. Nordström, and C. Tschudin, "Coping with communication gray zones in 802.11b based ad hoc networks," in *Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, Sep. 2002, pp. 49–55.

[178] W. Jiang and C. Zhang, "A portable real-time emulator for testing multi-radio MANETs," in *Proceedings of 20th IEEE/ACM International Symposium on Parallel and Distributed Precessing (IPDPS'06)*. IEEE Computer Society Press, Apr. 2006, p. 7.

[179] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET Simulation Studies: The Incredibles," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 9, no. 4, pp. 50–61, 2005.

[180] "The monarch project - wireless and mobility extensions to ns-2," http://www.monarch.cs.rice.edu/cmu-ns.html, accessed 2008-09-24.

[181] M. Lacage, "Ns-2 ieee 802.11 support," http://yans.inria.fr/ns-2-80211/, accessed 2008-09-24.

[182] "IEEE 802.11 in NS-2," Online at http://sarwiki.informatik.hu-berlin.de/Network_Simulator_ns2, last modified 2007-06-05, accessed 2008-09-24.

[183] *The ns Manual*, http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf, accessed 2008-09-29.

[184] T. S. Rappaport, *Wireless Communications: Principles and Practice*. Prentice Hall, Dec. 2001.

[185] M. Umlauft and P. Reichl, "Experiences with the ns-2 Network Simulator - Explicitly Setting Seeds Considered Harmful," *Wireless Telecommunications Symposium, 2007. WTS 2007*, pp. 1–5, April 2007.

A. Appendix

A.1. List of Parameters for the Server Selection Algorithm

We present a complete list of algorithm parameters as well as their default values in our implementation.

- Weight (Default: none - in our evaluation: uniformly distributed integral number between 0 and 63)
  The weight defines the capabilities of a node to become a server. A value of zero means that the node is unable to become a server. A node with a higher weight is better suited for the server role than a node with a lower weight.

- Discovery interval 1 (Default: 100 ms)
  The first discovery interval is used during the discovery phase of the server selection algorithm or anytime a client does not have a server. It determines that a node waits between the transmission of two successive announcement messages. During the discovery phase, the neighbour table is initially filled with data from the announcements of other nodes so this interval should be comparatively small. The actual delay between two announcement messages is varied by $\pm 20\,\%$ using a uniformly distributed random variable to reduce collisions in the network.

- Discovery interval 2 (Default: 500 ms)
  The second discovery interval is used in all other phases when the node itself is a server node or has determined its server. Here, changes in the network still need to be detected but is it advisable to reduce the overhead of announcement messages. Again, the interval is varied by $\pm 20\,\%$.

- Neighbour expire time (Default: 3 s)
  Each node maintains its own neighbour table in which it stores information about

neighbouring nodes. When no announcement message is received from a node for this amount of time, the node is removed from the neighbour list.

- minTx(Default: 3)
  MinTx is the number of announcement messages that the local node must be sent during discovery phase before it can proceed to the selection phase. The reason for this variable is to prevent a premature transition to the next phase in cases when announcement messages are lost early on. It should be defined as a trade-off between the required minimum converging time of the server selection algorithm and the time it takes to get good information about the network neighbourhood.

- maxTx(Default: 10)
  This value defines the maximum number of announcement messages that the local node sends before it is forced into selection phase. This value is used as a safeguard so that the algorithm always reaches the next phase even if a node keeps changing its announcement messages either abuseivly or by mistake.

- Expired clients timeout (Default: 10 s)
  When a client did not send a request to the server for this amount of time, it is removed from the server's list of clients. This list is used by the server to keep track of the total number of clients to see if it's server role is beneficial to the network. A server with no more clients besides itself automatically reverts back to the client role.

- Dropped server interval (Default: 30 s)
  When a node drops its server role because it has no remote clients anymore, it is prevented from assuming the server role again for this period of time. This prevents accidental oscillations of the server status of a node. A value of zero disables this feature.

A.2. Latency Measurement

In Chapter 5 we describe the analysis of our WLAN latency measurement the help of our modified WLAN driver. This experiment uses two notebook computers and the application 'ping' to analyse the delay behaviour of request and response packets on a wireless network. In this section, we will explain in more detail how the individual values were measured. In our WLAN driver we implemented the following three additional functions to record events as well as record additional data about them. For all events a high-precision time-stamp that uses the CPU's internal time stamp counter (TSC) is stored.

- The queuing event (queue)
  The queuing event triggers when a data frame is passed on by the WLAN driver from the local networking stack to the WLAN hardware. The hardware then queues the frame until the medium is free so it can be transmitted. The queuing event stores the IP address of sender and receiver as well as the type and sequence number of the ICMP packet to allow correlation later on. Additionally, the memory address of the data frame is stored to be able to associate the following transmission event with this particular data frame.

- The transmission event (send)
  The transmission event records the successful or unsuccessful sending of a data frame. This event is triggered by an interrupt of the WLAN hardware as soon as the transmission of a frame is complete. For a transmission event, a number of additional data is collected. First of all, the memory address of the data frame that was given by the WLAN hardware is stored to correlate this transmission with a previous queuing event. Also, data about the transmission itself like if it was successful or if an error occurred, the number of retransmissions if any, the transmission speeds and the signal strength of the received acknowledgement from the receiving station.

  The time that passes between a queuing event and a transmission event contains the contention phase, the time that it takes to transmit the frame including any retransmissions, and the inter-frame spaces as defined in the IEEE 802.11g protocol. By using the collected information, we can then calculate all transmission times and inter-frame spaces and subtract them from the time period. We define the remaining time as the queuing time of the data frame which comes from any contention phase or time periods when the medium was busy.

- The reception event (recv)
  The reception event is also triggered through an interrupt by the WLAN hard-

ware to tell the driver that a frame has been received successfully. For a reception event the signal strength, transmission rate, error status as well as IP addresses, type and sequence number of the ICMP packet are stored.

Data from the reception event of an response packet is used together with the transmission event of the corresponding request packet to determine the overhead by the kernel on the local machine. This value is calculated By comparing the round-trip time in the WLAN driver and of our ping application.

To explain our approach in detail, Figure A.1 presents an example of the measurement of a single ICMP request/response pair. We can see one node on the left and another on the right side which communicate through wireless LAN. Both nodes are shown as a simple networking stack. The black circles with white numbers show the sequence of actions and events which we will referer to below.

❶ Application sends request packet

The ping application sends an ICMP request packet and records the time t_{req} at which it submits the packet to the networking layer. By using t_{resp} later on, ping can calculate the round-trip of the packet pair. For our measurement, we have added high-precision timestamping to the ping application as to calculate the time delay inside the kernel. The networking stack converts the packet into a frame and sends it on to the WLAN driver. Our driver records another timestamp $t_{\mathrm{req-queue}}$ as it sends the frame to the WLAN hardware.

❷ WLAN driver queues request frame to hardware

The WLAN hardware stores the data frame for transmission until its clear channel assessment (CCA) function shows a free medium and all necessary WLAN protocol requirements (e.g a DIFS waiting time after each transmsission) has been fulfilled.

❸ Request frame is transmitted via WLAN to node #2

The request frame is transmitted over the wireless channel. The sender then waits from an acknowledgement from the receiving node. If this acknowledgment is not received the request frame is scheduled to be retransmitted.

❹ Request frame is received by node #2

When the request frame is received, the WLAN hardware triggers an interrupt to inform the driver that a new frame is available in its buffer. At this time, our WLAN driver triggers an reception event and records the time of arrival of the request frame $t_{\mathrm{req-recv}}$. Additional information like the received signal

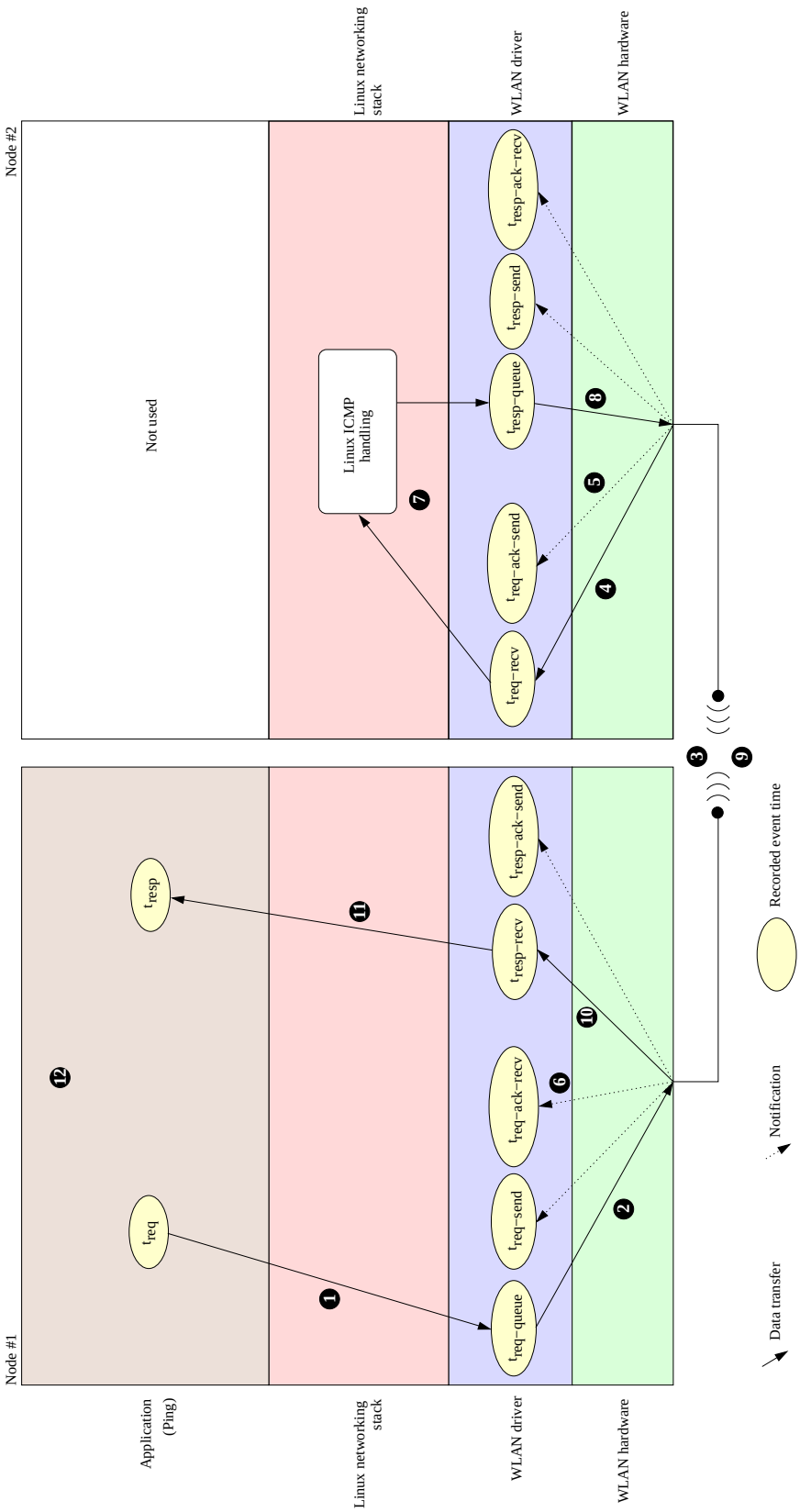


Figure A.1.: Event and time recording points in the latency measurement experiment

strength, the status of the checksum, the actual tranmsission speed and more are also available at this stage.

❺ Acknowledgement sent for request frame

The request frame is received by node #2 and the WLAN hardware automatically sends an acknowledgment back to node #1. This event is also recorded by the WLAN driver as $t_{\mathrm{req-ack-send}}$. Although a control frame is being sent out, the hardware creates a reception event for each acknowledgement. Acknowledgments are sent automatically by the hardware because of their hard time constraints, an acknowledgement has to be sent directly after a data frame has been received plus the SIFS waiting time. To cirumvent the fact that the WLAN driver has no prior knowledge of an acknowledgment, the event sent by the hardware is actual a reception event of its own acknowledgment.

❻ Acknowledgement received by node #1

Upon reception of the acknowledgment from node #2 the WLAN hardware triggers an interrupt to signal a transmission event. This time is recorded as $t_{\mathrm{req-send}}$. On node #1 the time between the queuing of the request frame $t_{\mathrm{req-queue}}$ and the transmission event $t_{\mathrm{req-send}}$ is the sum of the following items:

- The time that the node has to wait until the medium is free
- A DIFS waiting period and a contention phase in case the medium was busy with a prior transmission
- The transmission of the request frame
- A SIFS waiting period
- The transmission of the acknowledgment frame

In case of either the request frame or the acknowledgment was lost during transmission or a collision occured, the same items are repeated for a retransmission. Based on our timestamps and the data from the events we are able to calculate the theoretical duration of each item but the first one. Unfortunately, there is no off-the-shelf WLAN hardware that allows direct access to the CCA function or details of the contention phase. Therefore, we defined the contention and congestion time as the time measured between $t_{\mathrm{req-queue}}$ and $t_{\mathrm{req-send}}$ and substract our calculated time values for all transmissions and inter-frame spaces. Finally, we perform additional test to check the consistency of our results.

7 Handling the request packet

On the receiving node #2, the request frame is unpacked by the Linux networking stack and passed on to the functions that handle ICMP requests. In Linux like in most operating systems, ICMP is handeled directly inside the kernel without the need for any application program. It generates an ICMP response packet with the same sequence number as the request and send a response frame back to the WLAN driver for transmission.

8 Sending the response frame

The WLAN driver sends the response frame to the WLAN hardware and stores the queuing time $t_{\mathrm{resp-queue}}$. Again, a transmission event is sent by the hardware to the driver when the frame has been sent as well as notification that an acknowledgment has been sent. This procedure is the same as previously discussed in step 2.

9 Response frame is transmitted via WLAN to node #1

The respsonse frame is transmitted to node #1 which it sends an acknowledgement upon reception. Again, the frame may be retransmitted if either the frame or the acknowledgment get lost or if a collision occurs.

10 Reception of the response frame

After the WLAN hardware received the response frame it triggers an interrupt to inform the WLAN driver about this reception event. The WLAN driver stores the current time as $t_{\mathrm{resp-recv}}$. Also, notification of the corresponding acknowledgement is sent to the WLAN driver. While information about of acknowledgments is not needed to perform our latency analysis besides the fact that an acknowledgement has been sent and at which transmission speed, it provides interesting insight on how often acknowledgements were lost. When an acknowledgement is lost, the data frame itself was successfully received. But as the sending node does not receive an acknowledgment an unnecessary retransmit is triggered. While we did analyse this behaviour in detail, we did not include our results here as they in our case have had little impact on latency.

11 Recepetion of the response packet at the application

The received response frame is processed by the Linux networking stack and queued in the receiving socket buffer for the application. As soon as the application is scheduled to run by the operating system, it reads its buffer and timestamps the reception of the response packet (t_{resp}). The time between the reception of the packet at the WLAN driver ($t_{\mathrm{resp-recv}}$) and the reception at the

application (t_{resp}) is also added to the local kernel overhead. Because we did not use an application on node #2, we define the total kernel overhead in this scenario as two times the kernel overhead in node #1.

12 Round-trip time calculation

Finally, the application uses t_{req} and t_{resp} to determine the round-trip time of the request/response pair. Later, during the offline evaluation, we use all measured and all calculated values to perform serveral plausibility checks as to ensure the validity of our results.

A.3. Preliminary Evaluation - Additional Details

This section presents additional results from the preliminary evaluation in Chapter 5. The following Figures A.2, A.3, and A.4 show the individual results for latency, latency variation and packet loss for the discussed QoS mechanisms. They show that RTS/CTS adapation has a positive influence on all three values. Additionally and as previously mentioned in Chapter 5, priority queuing and rate control have a larger impact on packet loss. Here we also see that timeouts or early discard has a mentionable impact mainly because it generally reduces the network load in our scenario. Figures A.5 – A.8 show the effect that our quality of service methods have an low priority background traffic.

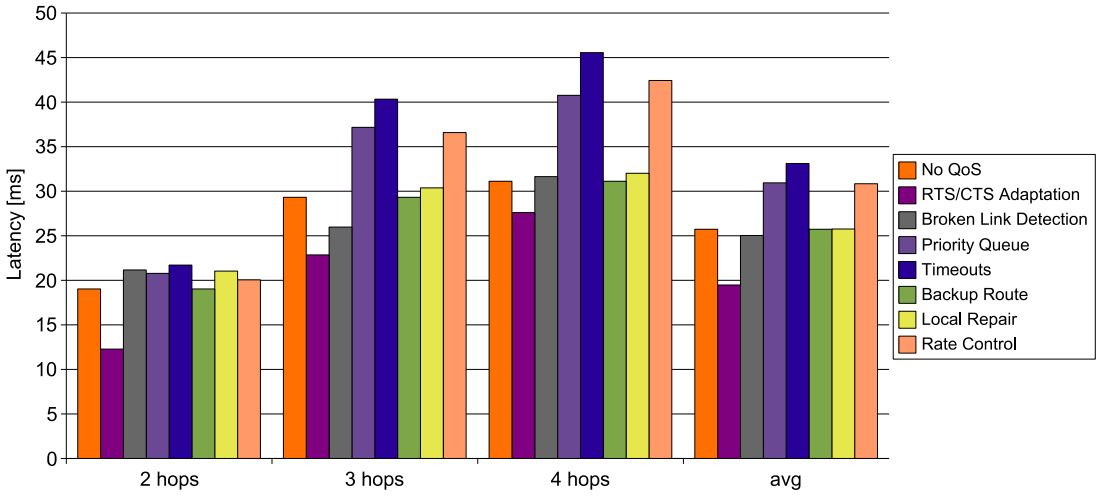


Figure A.2.: Average one-way latency for high priority traffic (individual impact, from [160])

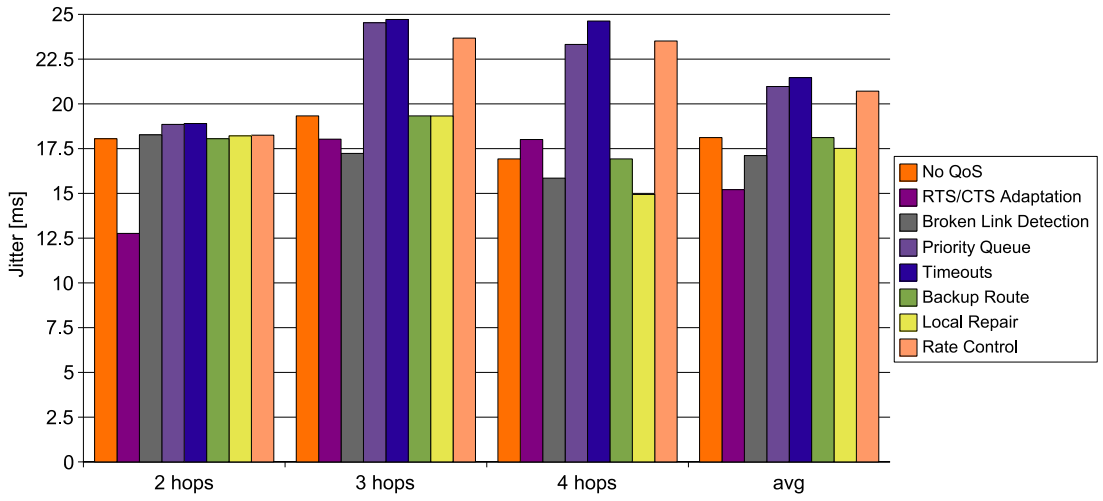


Figure A.3.: Average one-way delay variation for high priority traffic (individual impact, from [160])

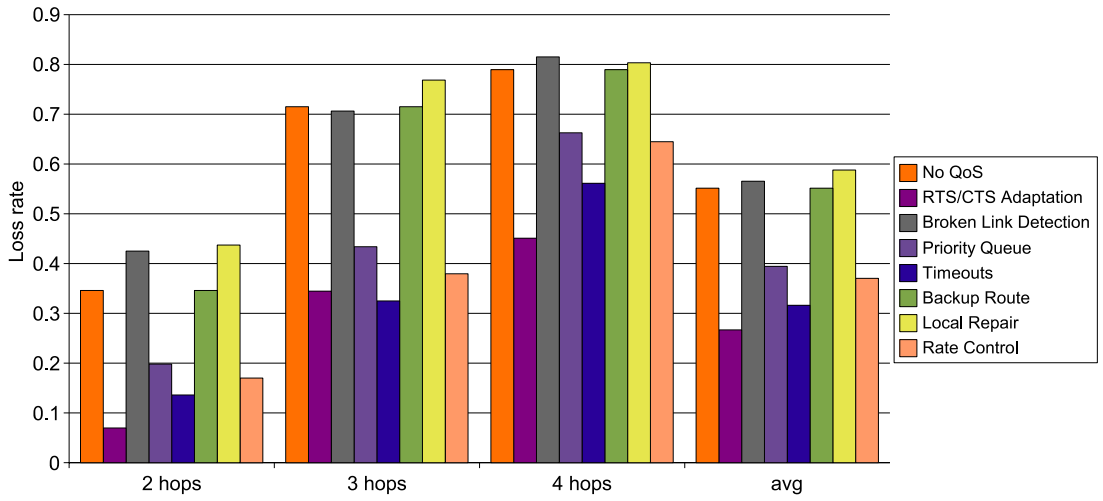


Figure A.4.: Average packet loss for high priority traffic (individual impact, from [160])

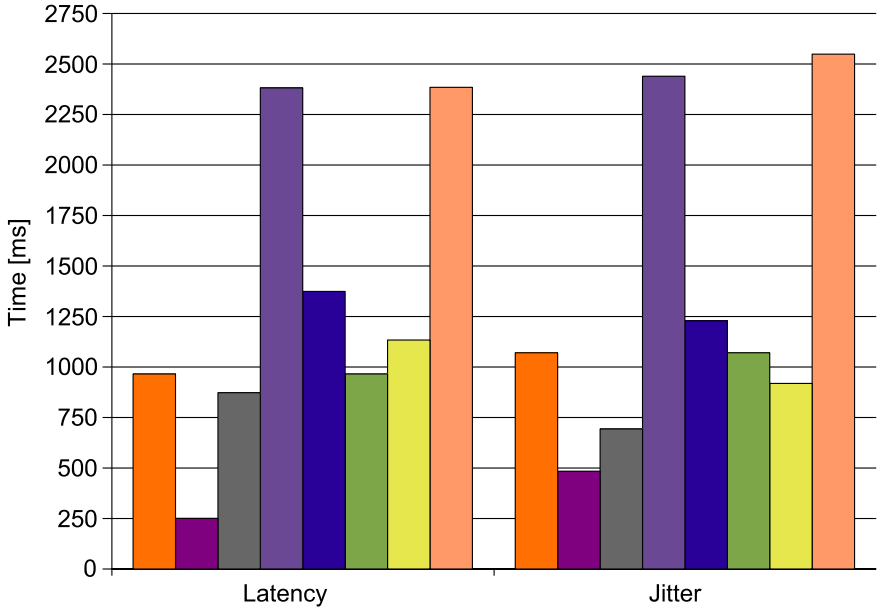


Figure A.5.: Average one-way latency for low priority traffic (individual impact, from [160])

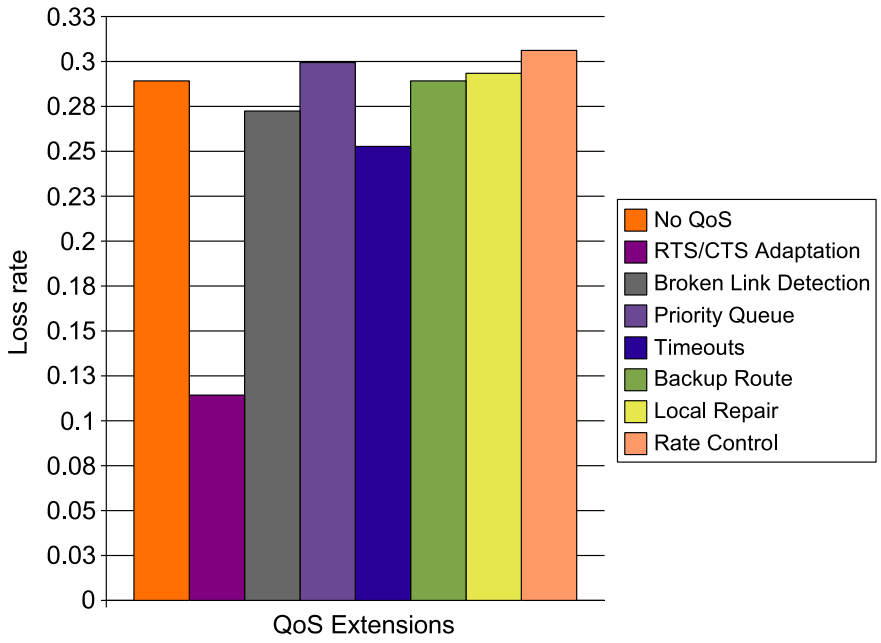


Figure A.6.: Average packet loss for low priority traffic (individual impact, from [160])

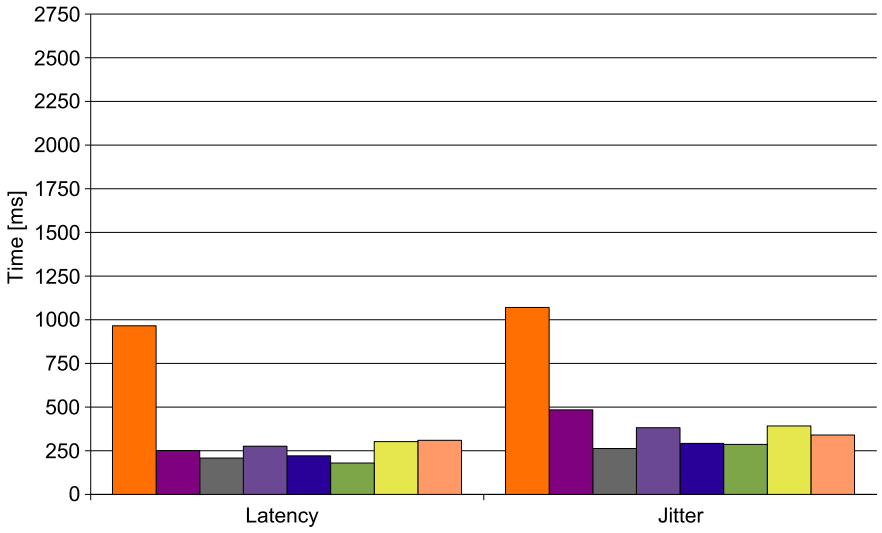


Figure A.7.: Average one-way latency for low priority traffic (combined impact, from [160])

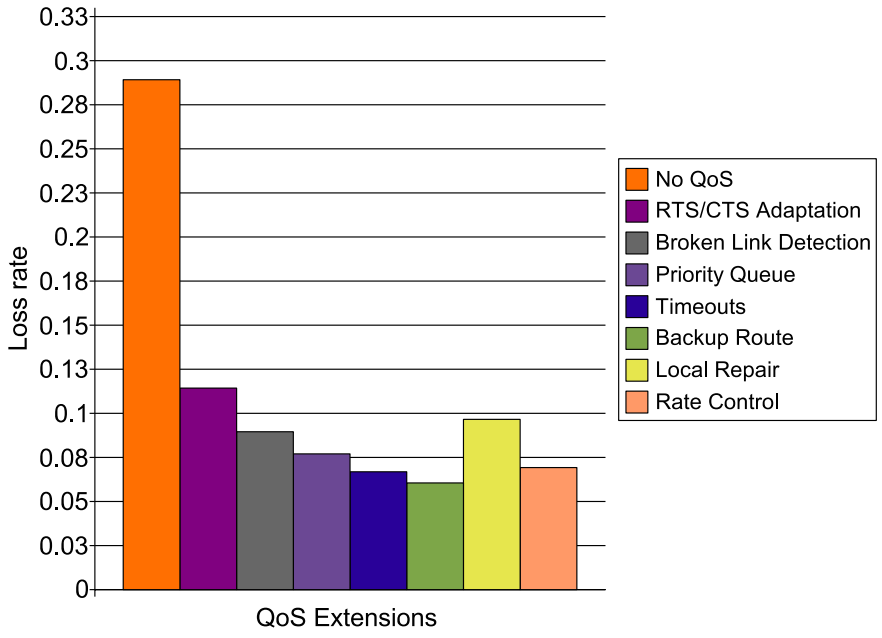


Figure A.8.: Average packet loss for low priority traffic (combined impact, from [160])